technische universität
dortmund

Diplomarbeit

Fehlerbehandlung in echtzeitfähigen Steueranwendungen

Dennis Nahberger
10. Mai 2012

Betreuer:
Prof. Dr. Peter Marwedel
Dipl.Inf. Andreas Heinig

Technische Universität Dortmund
Fakultät für Informatik
Lehrstuhl Informatik 12
Arbeitsgruppe Entwurfsautomatisierung für
Eingebettete Systeme
http://ls12-www.cs.tu-dortmund.de/daes/

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet, sowie Zitate kenntlich gemacht habe.

Dortmund, den 10. Mai 2012

Dennis Nahberger

Zusammenfassung

Zukünftige eingebettete Anwendungen werden stets mit der Notwendigkeit konfrontiert sein, Fehler, die im laufenden Betrieb auftreten, zu behandeln. Die zunehmende Miniaturisierung von integrierten Schaltkreisen begünstigt unvorhersehbare transiente Fehler, wie Veränderung übertragener Bitströme durch elektromagnetische Interferenzen. Diesen unabwendbaren Fehlern gegenüber stehen die nicht unbegrenzt vorhandenen Ressourcen zur Fehlerbehandlung. Zeitschranken müssen eingehalten werden und Redundanzen sind nur begrenzt. Um die vorhandenen Möglichkeiten effizient einsetzen zu können, hilft zusätzliches Wissen über die Art der Anwendung und die Fehler, die auftreten können. Dieses Wissen hilft dabei, zu entscheiden, mit welcher Priorität ein Fehler behandelt werden soll und ob es überhaupt einer Fehlerbehandlung bedarf. Ein Weg, Wissen über eine Anwendung zu gewinnen ist, Fehler, die in einer Anwendung entstehen können, zu identifizieren und danach zu klassifizieren, wie sie die Qualität der Leistung beeinträchtigen, die die Anwendung erbringen soll. Unter Verwendung von Robotern der Lego NXT Mindstorm Produktserie, können Beispiele für beschriebene eingebettete Steueranwendungen entwickelt werden. Ziel dieser Diplomarbeit ist es, eine Steueranwendung zu entwickeln, die durch die beabsichtigte Erzeugung von Fehlern untersucht wird und um eine Klassifikation der gefundenen Fehler zu erstellen.

Inhaltsverzeichnis

1 Einleitung

Bei der Entwicklung von Computersystemen muss davon ausgegangen werden, dass unvorhersehbare Fehler den Betrieb des Systems beeinträchtigen werden und diese Fehler behandelt werden müssen. Auf eingebettete Systeme trifft das ebenfalls zu, doch kommt hinzu, dass eingebettete Systeme stärker eingeschränkt sind in den verfügbaren Ressourcen zur Fehlerbehandlung, wie Rechenzeit oder Speichergröße oder Redundanzhardware. Methoden, die zum effizienteren Einsatz der verfügbaren Ressourcen beitragen, sind deshalb ein interessantes Thema zur Verbesserung von Fehlertoleranz in eingebetteten Systemen. In den folgenden Abschnitten des Einleitungskapitels werden Grundlagen vermittelt, die zum Verständnis der Problematik beitragen und zur Motivation für diese Diplomarbeit führen.

1.1 Ursachen für Fehler

Ein Computersystem wird mit dem Ziel entwickelt, dass es die Aufgabe, für die das System bestimmt ist, mit möglichst großer Performance bewerkstelligt. Das Streben nach der besten Performance bringt jedoch Nachteile mit sich. Kürzere Entwicklungszeiten und die zunehmende Verkleinerung von Gerätestrukturen, um Leistungssteigerung zu erzielen, begünstigt das Auftreten physikalischer Störungen. Die Verkleinerung von Schaltkreisen wird dadurch geschafft, dass mehr elektronische Bauteile auf der gleichen Chipfläche untergebracht werden. Mehr Bauteile auf der gleichen Fläche unterzubringen, setzt aber eine Verkleinerung der Bauteile wie Transistoren und möglichst kurze Leitungen voraus. Außerdem werden Leistungssteigerungen dadurch erzielt, dass die Bauteile höher betaktet werden oder die Betriebsspannungen verringert werden. Diese Faktoren bewirken, dass in Schaltkreisen mehr Störungen, wie das Übersprechen von Leitungen, auftreten und dass Schaltkreise anfälliger gegen externe Einflüsse, wie magnetische oder thermische Veränderungen werden. Sensoren sind ein gutes Beispiel für Schaltkreise, die durch ihre Messaufgabe in direktem Kontakt zur physikalischen Umgebung stehen und eine Quelle für Fehler sind. Typische Fehler, die durch die beschriebenen Störungen verursacht werden, sind Bitfehler, die im nächsten Abschnitt beschrieben werden.

1.2 Bitfehler

Wenn sich ein im Speicher abgelegter oder per Leitungsstrom übertragener Wert unbeabsichtigt auf eine Weise ändert, so dass sich eines seiner Bits von 0 nach 1 (oder umgekehrt) ändert, spricht man von einem Bitfehler. Wenn ein Fehler dieser Art eintritt, wird das Bit als gekippt bezeichnet und im Englischen spricht man von einem Bitflip. Ein Bitfehler kann weiterreichende Konsequenzen mit sich ziehen, wenn er Daten oder Befehle eines Programms verändert. Eine solche Änderung kann dazu führen, dass an den falschen Stellen des Speichers Daten abgelegt werden oder verfälschte Daten weiterverarbeitet und fehlinterpretiert werden. Wenn die Möglichkeit besteht, direkt Einfluss auf Speicherinhalte zu nehmen, lassen sich Bitfehler zu Untersuchungszwecken absichtlich erzeugen, wie es später in Abschnitt 4.2.6 beschrieben wird. Je nachdem mit welchen zeitlichen Abständen Fehler auftreten, lassen sich unterschiedliche Arten von Fehlern feststellen.

1.3 Arten von Fehlern

Man unterscheidet zwischen drei Arten von Fehlern: permanente Fehler, transiente Fehler und intermittente Fehler [1].

- Permanente Fehler sind nach dem Auftreten dauerhaft in einem System vorhanden. Wenn ein Chip durch Probleme bei der Herstellung eine defekte Komponente beinhaltet, ist bis zum Austausch der Komponente ein permanenter Fehler vorhanden. Falsche Designentscheidungen bei der Softwareentwicklung ebenfalls Fehler erzeugen, die bis zur Korrektur permanent sind.

- Transiente Fehler sind nur für begrenzte Zeitintervalle oder Zeitpunkte existent und verschwinden danach wieder. Wirkt eine kurzzeitige elektromagnetische Störung auf einen Chip ein, kann es in schlecht isolierten Leitungen zu Übertragungsfehlern führen, die nachlassen, sobald die Störung nachlässt.

- Intermittente Fehler treten in unregelmäßigen Abständen zueinander immer wieder auf. Thermische Auswirkungen, die durch hohe Prozessorlast verursacht werden, können regelmäßig auf physisch angrenzende Leitungen einwirken.

Da sich permanente Fehler nur durch die Korrektur der Fehlerursache beheben lassen, kann aus Sicht der Anwendung weniger unternommen werden, um auf diese Fehler zu reagieren. Transiente Fehler hingegen belasten nicht dauerhaft ein System. Ihre Auswirkungen können innerhalb einer Anwendung durch geeignete Fehlertoleranzmechanismen reguliert werden, was sie zu interessanten Objekten der Untersuchungen dieser Diplomarbeit macht. In welcher Weise Fehler Anwendungen beeinträchtigen können, wird im nächsten Abschnitt beschrieben.

1.4 Fehler in Anwendungen

Je nachdem, wo ein Fehler auftritt, kann er sich unterschiedlich auf eine Anwendung auswirken. Fehler, die die Ausführung eines Anwendungsprogramms verändern, indem Befehle verändert oder in der falschen Reihenfolge ausgeführt werden, sind ein Problem, welches allgemein jede Anwendung betreffen kann. Wird eine spezielle Anwendung untersucht, bieten sich Datenfehler zur Untersuchung an, die das Verhalten einer Anwendung ebenfalls verändern. Als Datenfehler sind alle Fehler zu verstehen, die Veränderung von Werten, die in Variablen gespeichert werden, zur Folge haben. Liefert ein Sensor einen fehlerhaften Messwert, der in einer Variablen zwischengespeichert wird, handelt es sich um einen Datenfehler. Im nächsten Abschnitt werden Methoden vorgestellt, die zur Untersuchung von Fehlern verwendet werden können.

1.5 Fehlersimulation und Fehlerinjektion

Die in den vorhergehenden Abschnitten vorgestellten Arten von Fehlern und der Effekte, die sie auf ein System haben, motiviert dazu, Fehler in Systemen weiter zu untersuchen.

Um Fehler in einem System untersuchen zu können, ist eine mögliche Methode die Verwendung von Fehlersimulation. Ein Systemmodell wird dahingehend verändert, dass es auf einen Fehler so reagiert, wie es das fehlerbehaftete System tun würde. Auf diese Weise kann die Fehlersimulation dazu verwendet werden, zu betrachten, welcher Fehler welches Maß an Auswirkungen mit sich zieht und ob Fehlertoleranzmechanismen die Auswirkungen reduzieren können. Um Aussagen über das Fehlerverhalten eines Systems mit Hilfe von Fehlersimulation treffen zu können, ist es notwendig, das System mit allen im Modell enthaltenen Fehlern zu simulieren und ein große Menge an Eingabedaten zu verwenden. Dadurch wird Fehlersimulation zu einem aufwendigen Verfahren bezüglich Zeit- und Rechenaufwand [2].

Wenn Fehlersimulation für ein System zu aufwendig ist oder kein Systemmodell vorhanden ist, ist Fehlerinjektion eine Alternative. Hierbei werden Fehler direkt in ein laufendes System eingebracht und das Verhalten des Systems beobachtet. Fehler lassen sich sowohl physikalisch in ein System injizieren, als auch über eine Software-implementierte Fehlerinjektion. Physikalische Fehlerinjektion kann unter anderem dadurch angewendet werden, dass einzelne Pins einer integrierten Schaltung beeinflusst werden oder ganze Elemente einer Schaltung durch die Bestrahlung mit Schwerionen manipuliert werden [3]. Ein weitere Möglichkeit, Fehler zu injizieren, ist die Manipulation eines Systems über seinen JTAG Port, der auf vielen integrierten Schaltungen standardmäßig zu finden ist. Über einen JTAG Port können direkt auf alle internen Register und Ports der CPU Lese- und Schreibzugriffe durchgeführt werden.

Falls die benötige Hardware für eine physikalische Fehlerinjektion nicht zur Verfügung steht, oder die physikalische Injektion die Untersuchungen am Testobjekt behindern, kann Software-basierte Fehlerinjektion verwendet werden. In diesem Fall werden Fehler aus der Anwendungssoftware heraus, z.B. über einen separaten Prozess, direkt in den Speicher des Systems geschrieben, um Fehler zu erzeugen.

1.6 Motivation

In den vorherigen Abschnitten wurde beschrieben, wie Fehler in Computersystemen entstehen und wie sie diese beeinträchtigen. Da Fehlerbehandlungsmethoden ein notwendige Eigenschaft von Computersystemen sind und es speziell in eingebetteten Systemen oft an den notwendigen Ressourcen mangelt, um alle Fehler rechtzeitig zu behandeln, sind Möglichkeiten um Ressourcen zu sparen oder die vorhandenen effizienter einzusetzen, sehr wünschenswert. Je mehr Informationen über die Fehler in einem speziellen System zur Verfügung stehen, desto effektiver können Fehler behandelt werden. Durch den Einsatz der vorgestellten Untersuchungsmethoden können diese Informationen gewonnen und dazu verwendet werden, Entscheidungen darüber zu treffen, welche Fehler welche Auswirkungen für ein System bedeuten. Eine Klassifikation der Fehler, die in einem System auftreten können, nach der Schwere ihrer Konsequenzen, ist ein hilfreiches Werkzeug bei der Entwicklung von Fehlertoleranz- und Fehlerbehandlungmaßnahmen.

1.7 Ziel der Diplomarbeit

Motiviert durch die zuvor beschriebenen Vorteile, die sich durch die Untersuchung von Fehlern ergeben, ist es das Ziel dieser Diplomarbeit, eine solche Untersuchung an einer Steueranwendung durchzuführen und zu beschreiben. Unter der Verwendung von Robotern der Lego NXT Mindstorm Produktserie muss eine Steueranwendung entwickelt werden, die für die Untersuchung auf Fehler geeignet ist. Das letztendliche Produkt dieser Diplomarbeit ist die Beschreibung der entstandenen Steueranwendung, die Darstellung der Ergebnisse, die sich aus der Untersuchung der Steueranwendung ergeben und eine Klassifikation der gefundenen Fehler.

1.8 Gliederung

Diese Arbeit gliedert sich in die folgenden Abschnitte:

In Kapitel 2 werden wissenschaftliche Arbeiten vorgestellt, die im Zusammenhang mit dieser Diplomarbeit stehen oder in der Diplomarbeit verwendet wurden. Es erfolgt eine Eingliederung dieser Arbeit in den Forschungskontext.

In Kapitel 3 wird beschrieben, welche Hardware, Software und Steuertechniken zur Erstellung der Steueranwendung verwendet wurden. Es wird der NXT Brick mit seinen zusätzlichen Sensoren und Motoren vorgestellt. Außerdem wird das verwendete Betriebssystem RTEMS und Steuertechniken für den NXT eingeleitet.

In Kapitel 4 folgt die Erstellung des Versuchsaufbaus mit einer Beschreibung der Steueranwendung und die Definition der Quality of Service Eigenschaften, die als Maßstab für die Experimente verwendet werden.

In Kapitel 5 wird die Fehlerklassifikation der Steueranwendung vorgestellt, die anschließend durch Experimente überprüft wird. Der in Kapitel 3 dargestellte Versuchsaufbau wird verwendet, um Messungen durchzuführen. Die Ergebnisse der Messungen werden dargestellt.

Kapitel 6 beinhaltet dann die Evaluation der Arbeit. Hier werden zunächst die Ergebnisse der Fehlerinjektion mit ihren Auswirkungen auf die Quality auf Service ausgewertet. Dann wird die Fehlerklassifikation unter Verwendung der Versuchsergebnisse überprüft.

In Kapitel 7 erfolgt dann eine Zusammenfassung der Arbeit und in Kapitel 8 werden im Ausblick Verbesserungsmöglichkeiten beschrieben.

2 Related Work

Diese Diplomarbeit ist im Rahmen des Forschungsprojekts ‘Flexible Error Handling for Embedded Real-Time Systems‘ (‘FEHLER‘) entstanden, das als Teil des Schwerpunktprogramms (SPP) 1500 ‘Dependable Embedded Systems‘ von der Deutschen Forschungsgemeinschaft (DFG) gefördert wird. Das ‘FEHLER‘-Forschungsprojekt arbeitet an der Realisierung von effizienten Fehlertoleranz-Methoden für zukünftige eingebettete Systeme. Motiviert ist dieser Forschungsschwerpunkt durch die zukünftig steigende Zahl von Hardwarefehlern, die durch den Trend zur Miniaturisierung bei der Herstellung von Halbleitertechnologien begünstigt wird. Aus dem ‘FEHLER‘-Projekt sind mehrere wissenschaftliche Arbeiten entstanden, von denen einige zur Erstellung dieser Diplomarbeit beigetragen haben.

Das Paper [4] (Heinig u. a. 2012) ist eine der Quellen, die diese Diplomarbeit motivieren. In dem Paper wird beschrieben, dass gewöhnliche Operationen elektronischer Geräte zunehmend durch die steigende Anzahl auftretender transienter Fehler beeinträchtigt werden. Da bei der Entwicklung von eingebetteten Systemen, speziell wenn es sich um Echtzeitsysteme handelt, meistens Restriktionen im Bereich Hardware- und Rechenzeitoverhead zu berücksichtigen sind, können in vielen Fällen nicht alle auftretenden Fehler behandelt werden, ohne die Restriktionen zu verletzen. Um die Menge der auftretenden Fehler auf ein behandelbares Maß zu reduzieren, empfiehlt es sich, ein System auf Fehler zu untersuchen und diese entsprechend ihrer Auswirkungen auf ein System und der Kosten ihrer Behandlung zu klassifizieren. Das gewonnene Wissen über die Fehlerklassen eines Systems kann anschließend dazu genutzt werden, um zu entscheiden, welche Fehler einer dringenden Behandlung bedürfen, welche Fehlerbehandlungen im Falle von Echtzeitschranken verzögert werden können und welche Fehler gänzlich zu ignorieren sind, da sie geringe oder gar keine Auswirkungen auf das System haben. Eine Echtzeitanwendung, die als gutes Beispiel für die Klassifikation von Fehlern dient, ist der H.264 Video Decoder. Durch die Ausnutzung von Anwendungswissen über den H.264, kann der Overhead, der zur Fehlerbehandlung benötigt wird reduziert werden.

Durch diese Arbeit ist das Interesse daran begründet, das Verfahren der Fehlerklassifikation auf eine Steueranwendung anzuwenden, wie es in dieser Diplomarbeit geschieht.

Die Paper [5] (Engel u. a. 2011) und [6] (Heinig u. a. 2012) sind ebenfalls aus dem ‘FEHLER‘-Forschungsprojekt hervorgegangen. Die Paper gehen der Problematik nach, dass Fehlertoleranz zukünftig von steigender Bedeutung für eingebettete Systeme sein wird, weil schrumpfende Halbleiterstrukturen und Versorgungsspannungen die Wahrscheinlichkeit auf transiente Fehler erhöht. Auf Grund der Tatsache, dass Hardwareredundanzen in eingebetteten System nur in geringem Maße zur Verfügung stehen, steigt

die Bedeutung von Software-basierten Methoden der Fehlerkorrektur. Es wird untersucht, wie Anwendungswissen genutzt werden kann, um Operationen und Variablen durch Annotation als ‘reliable‘ oder ‘unreliable‘ zu markieren. Eine als ‘unreliable‘ annotierte Operation erlaubt einen bestimmten Grad an Ungenauigkeit in der Ausführung, wohingegen ein Fehler in einer als ‘reliable‘ markierten Variable schwerwiegende Auswirkungen auf sein System haben kann. Diese Annotation hilft dabei zu entscheiden, welche Operationen oder Variablen bei der Nutzung von Ressourcen zur Fehlerkorrektur zu priorisieren sind und welche im Zweifelsfall auf Fehlerbehebung verzichten können. Dadurch trägt die Annotation dazu bei, automatische Fehlerklassifikation zu realisieren.

Die Aufteilung in beiden Extrema ‘reliable‘ oder ‘unreliable‘ wirft die Frage auf, ob es auch eine Graustufe zwischen den beiden Aufteilungen gibt, die sich zu untersuchen lohnt. Es ist von Interesse, ob eine ‘reliable‘ Komponente in einem System so verändert werden kann, dass sie im Fehlerfall weniger schwere Auswirkungen auf das System hat und im Fall der Entscheidung, welche von zwei ‘reliable‘ Komponenten die letzte verfügbare Ressource erhält, geringer priorisiert werden kann.

In der Bachelorarbeit [7] (Johnigk 2011) werden verschiedene Checkpointing und Rollback Recovery Verfahren untersucht, die eingesetzt werden, um die Fehlertoleranz von eingebetteten Systemen zu erhöhen. Durch die Sicherung von Prozesszuständen in sogenannten Checkpoints, welche zu bestimmten Zeitpunkten während des Programmablaufs angelegt werden, können erkannte Fehler behoben werden, indem ein Prozesszustand vor Auftreten des Fehlers wiederhergestellt wird. Die Wiederherstellung eines alten Prozesszustand, auch Rollback Recovery genannt, sorgt dafür, dass Fehler durch die erneute Ausführung einer vorher falschen Berechnung korrigiert werden können. Es wird ein Linker-Script verwendet, dass dafür sorgt, dass die Checkpointing relevanten Applikationsdaten in separate .appdata- und .appbss-Sektionen verlegt werden, während die übrigen Daten normal auf .data- und .bss-Bereich des Speichers aufgeteilt werden.

Dieses Linker-Script wird zur Fehlerinjektion in den Experimenten dieser Diplomarbeit verwendet. Das Linkerscript kann so verändert werden, dass gezielt die globalen Variablen einzelner Komponenten der Steueranwendung in die beschriebenen .appdata- und .appbss-Sektion verlegt werden. Um Fehler in bestimmte Variablen einer Komponente der Steueranwendung zur injizieren, wird das Linkerscript vor einem Experiment entsprechend abgeändert und die Variablen, in denen Fehler erzeugt werden sollen, global definiert. Über externe C-Variablen kann so auf die entsprechenden Bereiche des Speichers zugegriffen werden und Fehler verursacht werden.

Die Autoren Kiam Heong Ang u. a.(2005) bieten mit ihrer Arbeit [8] einen Überblick über Funktionalitäten und Tuning-Methoden von Proportional-Integral-Differential Reglern (PID-Regler). Es werden Software Pakete und kommerzielle Hardware vorgestellt, die zum Ziel beitragen, eine Art Plug-and-Play PID-Regler zu erstellen, der sich für viele Anwendungszwecke verwenden lässt und den aufwändigen Kalibrierungsprozess vereinfacht.

Die Beschreibungen dieser Arbeit und speziell die Erläuterungen über Kalibrierungsmethoden für PID-Reger haben zu der Entwicklung der Steueranwendung für den Lego NXT Mindstorm beigetragen.

3 Verwendete Software, Hardware und Steuertechniken

Um Untersuchungen an einer eingebetteten Steueranwendung durchführen zu können, muss zunächst eine Beispielanwendung entwickelt werden, die eine Steueraufgabe ausführt. Dazu wurden die Roboter der Lego NXT Mindstorm Produktreihe ausgewählt, weil sie sich hervorragend eignen, um Beispielanwendungen zu entwickeln, da bereits eine Vielzahl von Anwendungsmöglichkeiten veröffentlicht worden sind und Open Source Entwicklungsumgebungen, Programmierschnittstellen und Gerätetreiber für die NXT Sensoren und Motoren zur Verfügung stehen. Eine Portierung des Echtzeitbetriebssystem RTEMS für den Lego NXT Mindstorm ist bereits vorhanden, so dass sich dieses als Betriebssystem für die Steueranwendungen anbietet. Als Steueraufgaben für den verwendeten NXT wurden Linienverfolgung und Abstandswahrung ausgewählt, da die benötigten Sensoren zur Ausführung der Aufgaben im Umfang eines Lego NXT Bausatzes vorhanden sind. In diesem Kapitel erfolgt eine genauere Beschreibung der Software, Hardware und Steuertechniken, für die sich entschieden wurde, um die Beispielanwendung zu entwickeln.

3.1 LegoNXT Mindstorm

Für die Experimente in dieser Diplomarbeit, wurden zwei Lego Mindstorms der NXT Familie verwendet.

Abbildung 3.1: Der NXT Mindstorms Brick

Ein Mindstorm besteht aus dem programmierbaren NXT Brick (Abbildung 3.1), einem

Mikrocontroller mit Eingabe/Ausgabe-Interfaces. Die Hardware Spezifikation eines NXT Bricks sieht wie folgt aus [9]:

- Haupt Prozessor: Atmel® 32-bit ARM® processor, AT91SAM7S256 [10]
  - 256 KB FLASH
  - 64 KB RAM
  - 48 MHz
- Co-Prozessor: Atmel® 8-bit AVR processor, ATmega48
  - 4 KB FLASH
  - 512 Byte RAM
  - 8 MHz
- Bluetooth wireless Kommunikation: CSR BlueCoreTM 4 v2.0 +EDR System
- USB 2.0 Kommunikation: Full speed port (12 Mbit/s)
- 4 input ports: 6-wire interface supporting both digital and analog interface
- 1 high speed port, IEC 61158 Type 4/EN 50170 compliant
- 3 output ports: 6-wire interface supporting input from encoders

3.2 Sensoren und Aktuatoren

An jeden der vier Eingabe-Ports, über die der NXT-Brick verfügt, kann ein Sensor und an jede der drei Ausgabe-Ports ein Motor angeschlossen werden.

Abbildung 3.2: Lego Mindstorms Komponenten; von links nach rechts: Servo Motor, Licht Sensor und Ultraschall Sensor

Jeder Servo Motor (Abbildung 3.2 links) verfügt neben seinen regulären Motorenfähigkeiten zusätzlich über einen eingebauten Sensor, der unter anderem die Umdrehungen misst, die der Motor absolviert hat. Die gemessenen Umdrehungen können auf 1 Grad

genau abgefragt werden. Der Lichtsensor (Abbildung 3.2 mittig) verfügt über ein integriertes Infrarotlicht, welches aktiviert werden kann, falls die Umgebungsbeleuchtung nicht ausreichend für genaue Messungen ist. Er liefert Werte in einem Bereich von 0 (kein Licht) bis 1023 (sehr hell). Der Ultraschallsensor (Abbildung 3.2 rechts) sendet ein Signal aus und berechnet aus der Zeit zwischen dem Senden und dem Empfangen des Reflexionsignals den Abstand zu Objekten in seinem Sichtfeld. Der gemessene Abstand kann in cm abgefragt werden und beträgt maximal 255 cm. Es sind noch weitere Lego Mindstorm Sensoren, wie z.B. Farbsensor, Temperatursensor und Berührungssensor, verfügbar, die jedoch für die Experimente in dieser Arbeit nicht verwendet wurden und deshalb nicht weiter beschrieben werden.

3.3 RTEMS

RTEMS ist ein Open Source Betriebssystem für eingebettete Multicore- und Echtzeitsysteme [11]. Zum Entstehungszeitpunkt wurde RTEMS vom US-Militär zur Steuerung von Raketen entwickelt, doch mittlerweile wird es vom im Jahr 2001 gegründeten Steering Committee weiterentwickelt. Ziele der Weiterentwicklung sind es, RTEMS Unterstützung für neue Architekturen und Prozessor Modelle zu schaffen. Hauptsächlich wird RTEMS in Systemen eingesetzt, die besonders wenige Ressourcen zur Verfügung haben, weshalb der Aufbau von RTEMS besonders stark modular gehalten ist. Grundlegende Systemdienste wie Scheduling bzw. Dispatching liefert der RTEMS-Kernel, während weitere Komponenten wie unter anderem I/O-, Semaphore- oder oder Timer-Management optional verwendet werden können. Derzeit werden diverse Mikroprozessoren wie i386, 68K, PowerPC, SuperH, MIPS und SPARC unterstützt. Unterstützung für die ARM-Archtiketur, der der AT91SAM7S256 des Lego Mindstorm NXT angehört, ist ebenfalls vorhanden. Durch die Unterstützung offener Standards lassen sich Anwendungen mit RTEMS leicht auf andere Systeme übertragen, die die selben Standards unterstützen, was zur verbreiteten Verwendung von RTEMS beiträgt. Folgende Standards werden unter anderem von RTEMS unterstützt:

- POSIX 1003.1b API inklusive Threads
- RTEID/ORKID API
- TCP/IP Stack inklusive BSD Sockets
- uITRON 3.0 API
- GNU Toolset

Die Tatsachen, dass RTEMS eine Open Source Lösung ist und Unterstützung für ARM Mikrocontroller vorhanden ist, sind Gründe dafür, warum es eine gute Wahl als Betriebssystem für eine Steueranwendung auf dem NXT ist. Weitere Gründe die für die Verwendung von RTEMS sprechen, sind, dass es bereits eine Portierung für den Lego NXT Mindstorm ist gibt. Des weiteren ist das durch RTEMS ermöglichte Multitasking

hilfreich, um unterschiedliche Steueraufgaben des NXT auf separate Tasks aufzuteilen. Durch diese Aufteilung können Auswirkungen von Fehlerinjektion auf einzelne Teile der Anwendung mit Hilfe des verwendeten Linker Scripts besser untersucht werden.

3.4 nxtOSEK

Die Open Source Plattform nxtOSEK für den Lego Mindstorms NXT, ist die Verbindung aus den zwei Open Source Projekten leJOS NXJ und TOPPERS OSEK. TOPPERS OSEK wurde ursprünglich für Multitasking Features in eingebetteten Echtzeitanwendungen im Automobilbereich entwickelt. leJOS NXJ bietet eine C-Programmierschnittstelle zur Steuerung von NXT Sensoren und Motoren[12]. Die Gerätetreiber, die leJOS NXJ bereitstellt, lassen sich mit geringem Aufwand als Verbindungsstück zwischen einem RTEMS-Betriebssystem und der Lego Hardware integrieren. Die leichte Verwendbarkeit von leJOS NXT macht es zum idealen Kandidaten für die Steueranwendung des NXT, die im Rahmen dieser Diplomarbeit entstanden ist.

3.5 NQC und BricxCC

Der Lego NXT Mindstorm, der als Referenzroboter für die Experimente verwendet wird, soll mit der vorinstallierten Standardfirmware von Lego betrieben werden. Es wird also ein Methode benötigt, mit der die benötigten Teile des in C geschriebenen Steuerprogramms in eine Applikation portiert werden können, die von der Lego Firmware ausgeführt werden kann. Eine gute Lösung für die gesuchte Methode bietet Not Quite C (NQC). Not Quite C ist eine Programmiersprache mit C-ähnlicher Syntax, die für die Programmierung der Lego Mindstorms entwickelt wurde. Bis auf wenige Änderungen bei Sensor und Motorsteuerung lässt sich die Steueranwendung aus C leicht nach NQC portieren. Mithilfe des Bricx Command Center (BricxCC), einer unter Windows ausführbaren Entwicklungsumgebung für NQC, lassen sich die NQC Steueranwendungen compilieren, auf den NXT übertragen und direkt ausführen. Des weiteren unterstützt das Bricx Command Center, auch die Programmierung der Mindstorms in C++, Pascal, Forth und Java durch Verwendung der alternativen BrickOS, pbForth oder leJOS Firmwares.

3.6 PID-Regler

Um den NXT korrekt zu navigieren und zu regulieren, muss das Steuerprogramm regelmäßig Messwerte über die Umgebung einholen und auswerten. Typischerweise wird dafür nach einem bestimmten Intervall ein Messwert eingeholt, mit einem Sollwert verglichen und anschliessend Stellgrößen des Steuerprogramms so manipuliert, dass sich Mess- und Sollwert möglich weit annähern. Ein gute Methode, dies zu bewerkstelligen, ist ein PID-Regler. Die Regelgleichung eines PID-Reglers (Formel 3.1) besteht aus einem

Proportional- (P), einem Integral- (I) und einem Differentialanteil (D).

$$y = K_P * E + K_I * E_{SUM} + K_D * (E - E_{alt}) \tag{3.1}$$

Für den Proportionalanteil wird die Regelabweichung E mit einem Verstärkungsfaktor K_P multipliziert. Durch proportionale Veränderung der Stellgrößen kann jedoch nicht jede Regelschleife vollständig reguliert werden, da die Regelabweichung bestehen bleibt. Der Integralanteil summiert die Regelabweichungen über die Zeit auf und multipliziert die Summe mit einem Faktor K_I. Durch das Anwachsen der Abweichungssumme E_SUM werden die Stellgrößen stärker reguliert, je länger eine Regelabweichung vorliegt. Zur Berechnung des Differentialanteils wird ein Faktor K_D mit der Änderung der Regelabweichung multipliziert, was der Geschwindigkeit entspricht, mit der sich die Regelabweichung verändert. Durch den Differentialanteil kann schnell auf starke Veränderung der Regelabweichung reagiert werden. Die Faktoren K_P, K_I und K_D müssen je nach Anwendung speziell kalibriert werden.

3.7 Linienverfolgung

Unter einem linienfolgenden Roboter versteht man einen Roboter, der selbstständig einen Weg findet, einer vorgegebenen Strecke zu folgen, die gewöhnlich durch eine schwarze oder farbige, wenige Zentimeter breite, Linie auf weißem oder anders farbigem Untergrund dargestellt wird. Die Programmierung eines Roboters zur Linienverfolgung wird häufig als Unterrichtsaufgabe für Schüler oder Studenten verwendet, um Grundlagen über Robotersteuerung zu lehren. Es existieren jedoch auch reale Anwendungsgebiete für Linienverfolgung bei der Verteilung von Post innerhalb von Bürogebäuden oder Medikamenten in Krankenhäusern. Um es einem Roboter zu ermöglichen, einer Linie zu folgen, wird mindestens ein Licht- oder Farbsensor benötigt, der Messwerte über den Untergrund liefert, auf dem sich der Roboter befindet. Die Messwerte werden mit den bereits bekannten Werten über die Strecke verglichen, um zu entscheiden, ob auf einer Linie, abseits einer Linie oder an der Kante einer Linie gemessen wurde und zu entscheiden, in welche Richtung der Roboter gesteuert werden muss, um die Linie nicht zu verlieren. Vergleichswerte können entweder fest einprogrammiert werden oder vor Beginn der Linienverfolgung durch einen Kalibrierungslauf bestimmt werden. Je mehr Sensoren zur Verfügung stehen, desto einfacher wird die Aufgabe, da Sensorfehler leichter toleriert werden können und eine genauere Bestimmung der Position des Roboters auf der Linie berechnet werden kann.

Sind 2 oder mehr Sensoren vorhanden, können die Sensoren so am Roboter angebracht werden, dass sich mindestens ein Sensor auf jeder Seite der Linie befindet, wodurch frühzeitig auf ein bevorstehendes Verlassen der Linie reagiert werden kann.

Wird nur ein Lichtsensor verwendet, kann nicht entschieden werden, auf welcher Seite einer schwarzen Linie auf weißem Untergrund sich ein Roboter befindet, was bedeutet,

dass einer Kante der Linie gefolgt werden muss. Dadurch wird die korrekte Steuerung des Roboters fehleranfälliger, da, sobald die andere Kante einer Linie überfahren wird, der Roboter die korrekte Fahrtrichtung nicht mehr bestimmen kann. Der Vorteil der Verwendung von nur einem Lichtsensor ist, dass fehlerhafte Steueranweisungen leichter zu beobachten sind.

3.8 Simulation einer Beschleunigung / Bremsverzögerung

Wird den Motoren des NXT der Befehl gegeben, die Geschwindigkeit zu verändern, wird die Zielgeschwindigkeit praktisch unmittelbar erreicht. Selbst bei einer Beschleunigung von 0 Prozent auf 100 Prozent Leistung oder dem umgekehrten Bremsen, ist der Zeitraum, der zur Beschleunigung oder zum Bremsen benötigt wird, durch Beobachtung kaum wahrnehmbar. Die geringe Beschleunigungs- bzw. Bremszeit wird durch den Umstand verursacht, dass die Masse des Lego NXT Mindstorms gering ist im Verhältnis zu seiner Motorleistung. Wenn eine Abstandswahrung vom NXT durchgeführt werden soll, was für eine Untersuchung interessant ist, ist es deshalb sinnvoll, künstlich eine Beschleunigung oder Bremsverzögerung zu simulieren. Andernfalls könnte der NXT jederzeit mit maximaler Leistung fahren und müsste nur kurz vor einem Hindernis in wenigen Augenblicken die Geschwindigkeit auf Null reduzieren. Erst durch das Vorhandensein einer Beschleunigung und Bremsverzögerung ist es überhaupt sinnvoll, das Konzept der Abstandswahrung für den NXT umzusetzen, wobei es sich um einen wünschenswerten Untersuchungsgegenstand handelt. Die Simulation von Beschleunigung oder Bremsverzögerung lässt sich dadurch bewerkstelligen, dass jeder Zugriff auf die Motorengeschwindigkeit stets über eine weitere Schicht geleitet wird, die den Beschleunigungs- oder Bremsbefehl nur schrittweise entsprechend des gewünschten Wertes für Beschleunigung oder Bremsverzögerung umsetzt. Ist eine derartige Ebene vorhanden, können Beschleunigungs- oder Bremsverzögerungswerte vor Aktivierung des NXT oder auch während des laufenden Betriebs verändert werden, um externe Einflüsse wie Untergründe mit wenig Reibung, zu simulieren.

3.9 Abstandswahrung

Ein linienfolgender Roboter verwendet Abstandswahrung, wenn er Umgebungsdaten mit einem Sensor misst, um mögliche Hindernisse, wie andere Roboter, die sich auf der Strecke befinden, zu entdecken und seine Steuerung so anpasst, dass ein Kollision mit dem Hindernis vermieden wird. Als Sensor für Abstandswahrung kann ein Ultraschallsensor, wie der Lego Mindstorms Ultraschall Sensor, verwendet werden, welcher die Distanz zu Hindernissen in seinem Sichtfeld mit Hilfe von Ultraschall berechnen kann. Wird ein Hindernis entdeckt, muss die Geschwindigkeit der Motoren so angepasst werden, dass der Roboter rechtzeitig vor dem Hindernis zum Stehen kommt. Erster Schritt ist die

Berechnung des Bremswegs, um sie mit der Distanz möglicher Hindernisse zu vergleichen. Zur Berechnung des Bremswegs S_{B} wird die aktuelle Geschwindigkeit V_{A} und die geltende Bremsverzögerung A_{B} benötigt. Die Berechnung erfolgt wie in Formel 3.2.

$$S_B = (V_A)^2/(2 * A_B) \tag{3.2}$$

Überschreitet der Bremsweg den Abstand zu einem Hindernis, muss die Geschwindigkeit so angepasst werden, dass der Bremsweg wieder unter den Hindernisabstand reduziert wird. Zur Berechnung Zielgeschwindigkeit V_{B}, wie in Formel 3.3 dargestellt, wird die Distanz zum Hindernis S_{D} und die geltende Bremsverzögerung benötigt.

$$V_B = \sqrt[2]{(S_D * 2 * A_B)} \tag{3.3}$$

Bei den Berechnungen, die zur Abstandswahrung notwendig sind, gilt es zu beachten, dass ein Sensor eine gewisse Zeit benötigt, um einen Abstand zu ermitteln und möglicherweise Messungenauigkeiten vorliegen, so dass der Bremsweg um einen zusätzlichen Wert ergänzt werden sollte.

4 Definition der Quality of Service

In diesem Kapitel wird der Versuchsaufbau beschrieben, der verwendet wird, um die im nächsten Kapitel beschriebene Fehlerklassifikation experimentell zu überprüfen. Beschrieben wird die entwickelte Steueranwendung für den NXT und wie sie die Quality of Service beeinflusst, die als Maßstab verwendet wird, um Fehler zu klassifizieren.

4.1 Versuchsaufbau

Der Versuch besteht aus einer Strecke zur Linienverfolgung, auf der beide Roboter auf die Startmarkierungen A und B aufgesetzt werden, welche gleichzeitig auch die Zielmarkierungen sind. Startmarkierung A befindet sich dem Linienverlauf folgend, eine Roboterlänge addiert um einen Sicherheitsabstand direkt vor Startmarkierung B. Der Sicherheitsabstand ergibt sich aus dem Programm zur Abstandswahrung, dass in Abschnitt 4.2.4 beschrieben wird. Auf Startmarkierung B wird der NXT aufgesetzt, in dessen Programm Fehler injiziert werden. Im weiteren Verlauf der Diplomarbeit wird dieser Roboter als der Fehler-NXT bezeichnet. Startmarkierung A ist die Position, auf der der Roboter ohne Fehlerinjektion aufgesetzt wird, welcher des weiteren als Hindernis-NXT bezeichnet wird.

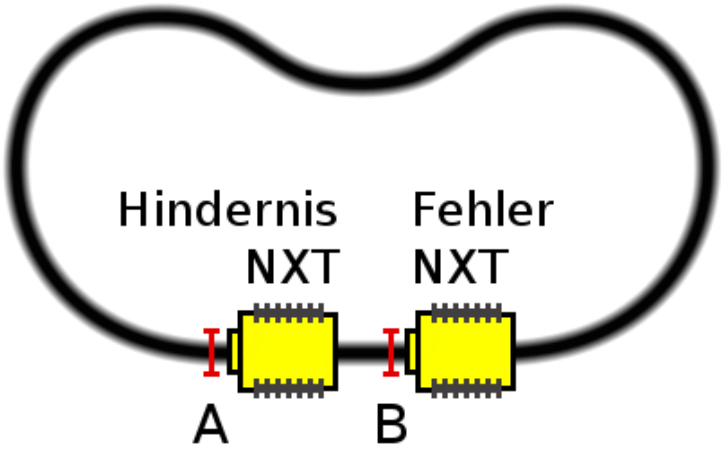

Abbildung 4.1: Versuchsaufbau

Zu Beginn jedes Versuchs werden die Programme beider Roboter gleichzeitig gestartet. Der Hindernis-NXT hat die Funktion, als bewegliches Hindernis zu agieren, auf das der Fehler-NXT zu reagieren hat. Der Hindernis-NXT folgt dabei dem Linienverlauf mit der gleichen Maximalgeschwindigkeit wie der Fehler-NXT, variiert jedoch seine

aktuelle Geschwindigkeit, um die Abstandswahrung des Fehler-NXT anzusprechen. Die Aufgabe des Fehler-NXT besteht darin, der Linie so schnell wie es sein Programm zulässt, zu folgen, dass bedeutet mit möglich wenig Steuerungsfehlern oder gar kompletten Verlieren der Linie und dabei den Sicherheitsabstand zum Hindernis-NXT einzuhalten. In diesem Versuchsaufbau wird die Zeitdifferenz zwischen dem Zeitpunkt des Eintreffens vom Hindernis-NXT auf Zielmarkierung A und dem Zeitpunkt des Eintreffens des Fehler-NXT auf Zielmarkierung B gemessen.

Soll statt der Zeitdifferenz die Rundenzeit betrachtet werden, wird der Versuchsaufbau so abgeändert, dass der Fehler-NXT alleine auf der Versuchsstrecke aufgesetzt wird und die Zeit zwischen Versuchsbeginn und dem Erreichen der Zielmarkierung gemessen wird.

4.2 Fehler-NXT

In diesem Abschnitt wird der Fehler-NXT und das Programm beschrieben, welches während der Versuche auf dem Fehler-NXT ausgeführt wird. Damit der Fehler-NXT Linienverfolgung und Abstandswahrung durchführen kann, sind ein NXT Licht Sensor und ein NXT Ultraschall Sensor an den Brick angeschlossen. Außerdem verfügt der Fehler-NXT über zwei Motoren, um die Räder anzutreiben. In Abbildung 4.2 ist der Fehler-NXT zu sehen.

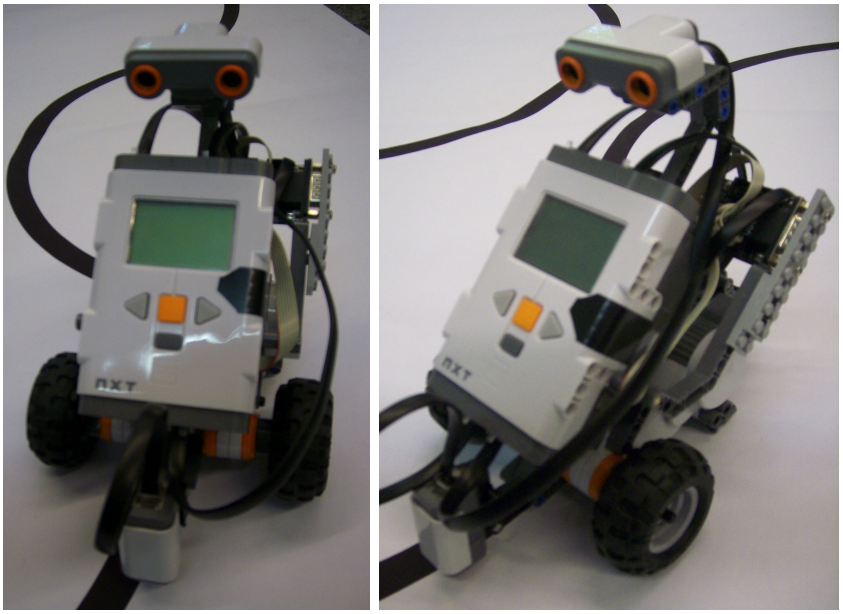

Abbildung 4.2: Der Fehler-NXT

Das Programm für die Versuche besteht aus fünf Tasks zur Steuerung des NXT, die über globale Variablen miteinander kommunizieren, und einer Task, welche die Fehlerinjektion durchführt. Die Aufteilung der Steueraufgaben auf fünf separate Task ist erfolgt, damit dem zur Fehlerinjektion verwendeten Linker-Script auf diese Weise gezielt Fehler in die einzelnen Komponenten der Steueranwendung injiziert werden können.

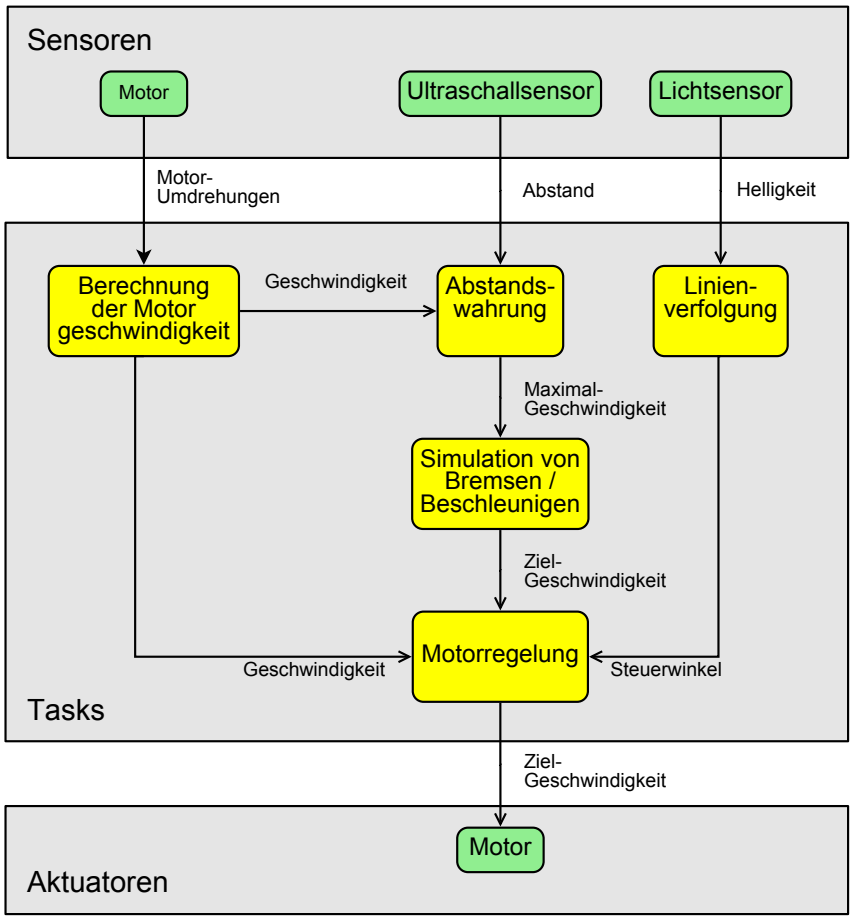

Abbildung 4.3: Programmaufbau und Taskkommunikation

Programmaufbau und Task-Beziehungen werden in Abbildung 4.3 dargestellt, die Beschreibung der einzelnen Task erfolgt in den nächsten Unterabschnitten.

4.2.1 Task zur Berechnung der Motorgeschwindigkeit

Innerhalb der Task zur Berechnung der Motorgeschwindigkeit wird der Umdrehungssensor, der in jeden NXT Servo Motor verbaut ist, abgefragt. Mit Hilfe der Informationen über den Radumfang der Lego Räder und der Umdrehungen seit der letzten Messung, kann die zurückgelegte Strecke in Millimetern und somit auch die Geschwindigkeit des NXT berechnet werden. Die Motorgeschwindigkeit wird für jedes Rad seperat berechnet und global zugänglich gemacht.

4.2.2 Task zur Regelung der Motorgeschwindigkeit

Jeder NXT Servo Motor lässt sich steuern, indem ein Leistungswert an den Motor übertragen wird. Diese Leistungswerte sind aus einem Wertebereich von 0 (keine Motorleistung) bis 100 (maximale Motorleistung). Vorteilhaft ist es, wenn dem Motor direkt eine Geschwindigkeit und nicht nur ein Leistungswert übertragen werden kann, da abhängig vom Ladestatus des Akkus oder anderen externen Einflüssen wie Untergründen mit verschiedenen Reibungswiderständen, ein Leistungswert zu unterschiedlichen Geschwindigkeiten des NXT führen kann. Die Übertragung von einem internen Geschwindigkeitswert in einen Leistungswert für die NXT Servo Motoren, findet in der Task zur Regelung der Motorgeschwindigkeit statt. Hierzu wird für jedes der beiden Räder ein PID-Regler verwendet, der die aktuelle Motorgeschwindigkeit, welche in der Task zur Berechnung der Motorgeschwindigkeit erzeugt wird, an die gewünschte Zielgeschwindigkeit anpasst. Wie die gewünschte Geschwindigkeit auf die beiden Motoren verteilt wird, wird unter Verwendung des Steuerwinkels berechnet, welcher in der Task zur Linienverfolgung erzeugt wird.

4.2.3 Task zur Linienverfolgung

Aufgabe der Task zur Linienverfolgung ist es, den Lichtsensor abzufragen und anhand der gemessenen Lichtwerte den Richtungswinkel festzulegen, mit dem der NXT gesteuert wird. Zur Berechnung des Steuerwinkels wird ein PID-Regler verwendet, der gemessene Lichtwerte mit Referenzwerten vergleicht, um die Position auf der Linie zu bestimmen. Die Referenzwerte über die hellsten und dunkelsten Lichtwerte die auf der Strecke gemessen werden können, werden vor Versuchsbeginn kalibiriert und fest kodiert, da sie während der Versuchsdurchführung konstant sind.

4.2.4 Task zur Abstandswahrung

In der Task zur Abstandswahrung wird in regelmäßigen zeitlichen Abständen der angeschlossene NXT Ultraschallsensor abgefragt, um mögliche Hindernisse zu entdecken. Gemessene Abstandswerte werden unter Verwendung der in Kapitel 3.9 vorgestellten Formeln umgerechnet, um die erlaubte Maximalgeschwindigkeit zu erhöhen oder zu verringern. Werden Abstandswerte während des Betriebs auf der Versuchsstrecke beim Ultraschallsensor abgerufen, kommt es in seltenen Fällen vor, dass falsche Werte ermittelt werden, die sehr stark von korrekten Messwerten abweichen und dem maximal möglichen Messwert entsprechen. Um diese Fehler auszugleichen, werden Messwerte in einem sortierten Array zwischengespeichert. Wird ein Abstandswert benötigt, wird stattdessen der Median der Messwerte verwendet. Die Verwendung eines Durchschnittswerts zum Ausgleich von Messfehlern funktioniert an dieser Stelle nicht, da die falschen maximalen Messwerte den Durchschnitt zu stark beeinflussen und eine zu hohe Maximalgeschwindigkeit herbeiführen.

4.2.5 Task zur Simulation von Beschleunigung und Bremsverzögerung

Um das Vorhandensein einer Beschleunigung und einer Bremsverzögerung zu simulieren, darf eine Änderung der Geschwindigkeit nicht unmittelbar an die Task zur Regelung der Motorgeschwindigkeit weitergeleitet werden. Wird die erlaubte Maximalgeschwindigkeit von der Task zur Abstandswahrung erhöht oder verringert wird die Zielgeschwindigkeit schrittweise und entsprechend der festgelegten Werte für Beschleunigung und Bremsverzögerung angepasst,um stets mit der maximal möglichen Geschwindigkeit zu fahren, ohne einen Zusammenstoß mit Hindernissen zu riskieren. Die Task zur Simulation von Beschleunigung und Bremsverzögerung ist von den Experimenten unter Einwirkung von Fehlerinjektion ausgeschlossen, weil die Task dazu dient, die physikalische Einwirkung von Beschleunigung und Bremsverzögerung zu simulieren, welche ohne Simulation zu gering wäre, um Abstandswahrung umzusetzen.

4.2.6 Task zur Fehlerinjektion

Die Fehlerinjektion zur Untersuchung von Fehlerauswirkungen wird von dieser Task aus durchgeführt. Es werden Fehler in alle global definierten Variablen einer Task injiziert, die vor der Kompilierung des Steuerprogramms im verwendeten Linker-Scipt festgelegt wird. Durch die Verwendung des Linker-Scripts befinden die sich alle global definierten Variablen der Task, die auf Fehler untersucht wird, zusammenhängend in separaten .appdata- und .appbss-Sektionen des Speichers. Auf die Sektionen kann während der Programmausführung über externe Variablen zugegriffen werden.

Die Fehlerinjektion soll sich so verhalten, dass Fehler über den gesamten Speicher injiziert werden können, obwohl sie nur im Data- und Bss-Bereichen tatsächlich durchgeführt werden soll. Dazu wird ein Zufallsbyte aus dem gesamten Speicher ausgewählt, doch nur wenn sich das Byte in dem Speicherbereich befindet, in dem Fehlerinjektion gewünscht ist, wird ein Fehler injiziert. Der Vorteile dieser Methode sind, dass nur die Fehler erzeugt werden, die tatsächlich untersucht werden sollen und dass Fehler in unregelmäßigen zeitlichen Abständen auftreten.

Wenn Fehler mit regelmäßigen Verzögerungen injiziert werden, besteht die Gefahr, dass sich eine Steueranwendung, welche auch in regelmäßigen Abständen bestimmte Operationen ausführt, bei jeder Injektion im gleichen oder einem ähnlichen Zustand befindet. Die Auswirkungen von Fehlern auf andere Programmzustände sind dann nicht zu untersuchen, weil sie nicht auftreten. Wird zum Beispiel jedes mal genau dann in einer Variable ein Datenfehler verursacht, wenn unmittelbar darauf ein neuer Sensorwert abgerufen wird, mit dem der Variableninhalt überschrieben wird, wirkt sich der Fehler nicht auf die Programmausführung aus.

Wenn eine Fehlerinjektion ausgeführt werden soll, wird im ausgewählten Byte ein Zufallsbit ausgewählt und geflippt. Um festzulegen, mit welcher Rate Fehler erzeugt werden, wird die Größe des Speicherbereichs, in den Fehler injiziert werden sollen, und die Größe des restlichen Speichers berücksichtigt. Sollen exemplarisch in 10% des Speichers Fehler mit einer Rate von einem Fehler pro Sekunde erzeugt werden, werden 10

Zufallsbytes pro Sekunde ausgewählt. Das führt dazu, dass durchschnittlich ein Byte in den 10% des Speicher liegt und eine tatsächliche Injektion durchgeführt wird. Wird im weiteren Verlauf der Diplomarbeit von einer Injektionsrate von einem Fehler pro Sekunde gesprochen, ist darunter durchschnittlich eine tatsächlich ausgeführte Injektion pro Sekunde zu verstehen.

4.3 Hindernis-NXT

Der Hindernis-NXT ist mit einem NXT Licht Sensor zur Linienverfolgung und zwei Motoren zum Antrieb der Räder ausgestattet. In Abbildung 4.4 ist der Hindernis-NXT und ein Reflektor für Schall abgebildet, der am Heck des Hindernis-NXT angebracht ist. Während der Testphase der Versuchsdurchführung wurde festgestellt, dass der Ultraschallsensor des Fehler-NXT den Abstand zum Hindernis-NXT schlecht bestimmen kann. Als ein Grund dafür wurde die Höhe angenommen, in der der Ultraschallsensor angebracht werden muss. Außerdem wurde festgestellt, dass glatte Hindernisse, die senkrecht zum Linienverlauf stehen, ebenfalls nicht vom Fehler-NXT erkannt werden. Aus den genannten Gründen wurde ein runder Reflektor konstruiert, der die Höhe des Hindernis-NXT vergrößert. Mit Hilfe des Reflektors ist der Fehler-NXT in der Lage, den Hindernis-NXT wahrzunehmen.

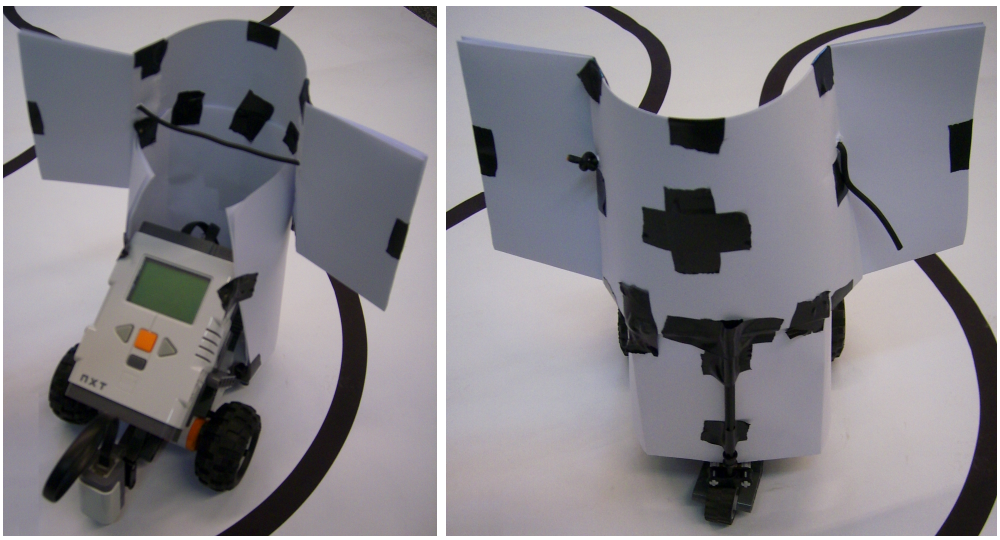

Abbildung 4.4: Der Hinderniss-NXT

Auf dem Brick des Hindernis-NXT ist die Firmware installiert, die von Lego standardmäßig für den NXT Brick vorgesehen ist. Durch die Verwendung der Standardfirmware kann der Hindernis-NXT neben seinen in Abschnitt 4.1 beschriebenen Aufgaben auch zum Vergleich mit der Leistung des Fehler-NXT verwendet werden. Um ein Steuerungsprogramm zu schreiben, welches von der Lego Software ausgeführt werden kann, wird das Bricx Command Center verwendet. Mit dem BricxCC kann das Steuerprogramm des Fehler-NXT, bis auf geringe Änderungen bei der Sensor- und Motoransteuerung,

auf den Hindernis-NXT übertragen werden. Zwei weitere Änderungen sind außerdem vorzunehmen. Die Task zur Abstandswahrung wird nicht mit in das Programm des Hindernis-NXT übernommen, weil sie für die Aufgaben des Hindernis-NXT nicht benötigt wird. Bei der zweiten Änderung handelt es sich um eine zusätzliche Funktion, die nur der Hindernis-NXT benötigt. Um als bewegliches Hindernis zu agieren, auf welches der Fehler-NXT zu reagieren hat, muss sich die aktuelle Geschwindigkeit, mit der sich der Hindernis-NXT fortbewegt, während eines Experiments ändern. Zu diesem Zweck wurde dem Programm ein Skript hinzugefügt, nach welchem sich die Veränderungen der Geschwindigkeit richten. Die Geschwindigkeiten und die Zeiträume zwischen den Änderungen der Geschwindigkeiten im Skript wurden zufällig generiert. Eine zufällige Generierung des Zeitplans während jeder Versuchsdurchführung kommt nicht Frage, um sicherzustellen, dass die Versuchsbedingungen in jedem Experiment konstant sind.

4.4 Quality of Service

Um die Leistung des Steuerungsprogramms des Fehler-NXT und die Auswirkungen von Fehlern untersuchen zu können, wird ein Maßstab benötigt, um die Güte der Steueranwendung zu bewerten. Als Maßstab wird eine Quality of Service (QoS) definiert. Die drei minimalen Qualitätsanforderungen an die Steueranwendung sind:

- Q_L Kein Verlust des Linienverlaufs
- Q_C Keine Kollision mit dem Hindernis-NXT
- Q_D Das Erreichen der Zielmarkierung vor Ende der Maximalzeit

Wird eine der minimalen Qualitätsanforderungen nicht erfüllt, ist die gesamte Quality of Service nicht mehr gewährleistet.

$$Q = \begin{cases} 1, & \text{Anforderung erfüllt} \\ 0, & \text{Anforderung nicht erfüllt} \end{cases} \tag{4.1}$$

Die vierte Qualitätseigenschaft (Q_T) ist die gemessene Zeit bei der Versuchsdurchführung (T_{NXT}). Die Differenz (T_Δ) der gemessenen Zeit (T_{NXT}) und der Minimalzeit (T_{MIN}) wird im Verhältnis zur Maximalzeit (T_{MAX}) betrachtet, wobei die Minimalzeit durch die Rundenzeit des Hindernis-NXT bestimmt wird, da der Hindernis-NXT die Rundenzeit des Fehler-NXT nach unten begrenzt. Als Maximalzeit wird die doppelte Minimalzeit festgelegt. Um den berechneten Werte für Q_T in ein gut zu betrachtendes Format zu überführen, wird er so umgeformt, dass sich ein QoS Wert in einem Wertebereich von 0 bis 1 befindet, wobei der Wert 1 der maximalen QoS entspricht.

$$Q_T = (1 - \frac{(T_{NXT} - T_{MIN})}{T_{MAX}}) \tag{4.2}$$

Der Fehler-NXT leistet genau dann seine maximale Quality of Service (QoS_{MAX}), wenn er keine der minimalen Qualitätsanforderungen verletzt und dabei die Minimalzeit benötigt, um seine Steueraufgabe auszuführen, also seine Zielmarkierung zu erreichen.

$$QoS = QoS_{MAX} \quad , f\ddot{u}r \quad T_\Delta \to 0 \tag{4.3}$$

Die QoS berechnet sich also wie folgt:

$$QoS = Q_L * Q_C * Q_D * Q_T \tag{4.4}$$

Wird der Versuchsaufbau zur Bestimmung der Rundenzeit des Fehler-NXT gewählt, ist die Minimalzeit (T_{MIN}) gleichzusetzen mit der minimal benötigten Rundenzeit des Fehler-NXT, die ohne den Einfluss von Fehlern gemessen wird.

5 Fehlerinjektion und Fehlerklassifikation

In diesem Kapitel wird die Erstellung einer Fehlerklassifikation für die Steueranwendung des Fehler-NXT beschrieben und wie die Fehlerklassifikation durch Verwendung von Fehlerinjektion experimentell überprüft wird.

5.1 Erstellen der Fehlerklassifikation

Datenfehler in einer Steueranwendung können je nach Zeitpunkt oder Ort des Auftretens unterschiedlich schwere Auswirkungen mit sich ziehen. Wird im Fall der Fehler-NXT Steueranwendung ein gespeicherter Sensorwert durch einen Fehler verändert, doch unmittelbar darauf und noch bevor der Sensorwert weiterverarbeitet wird, durch einen neuen Sensorwert überschrieben, zeigen sich keine Auswirkungen auf das Verhalten des NXT. Ein Leistungswert für einen Motor, der durch einen Fehler verändert wird, kann zu einer drastisch veränderten Geschwindigkeit oder Steuerrichtung führen, was zu einem Verlust des Linienverlaufs führen kann und somit die Quality of Service massiv beeinflusst. Dadurch, dass Fehler die Quality of Service auf unterschiedlich schwere Art und Weise beeinflussen, ist es sinnvoll, Kategorien zu erstellen, die Auskunft darüber geben, wie schwer ein Fehler die Quality of Service beeinflusst. Diese Kategorien können später dabei helfen, zu entscheiden, welche Fehler bei der Fehlerkorrektur höhere Priorität genießen sollten.

Bei einer veränderten Variable, die eine Division durch Null zur Folge hätte und somit Programmterminierung bewirkt, handelt es sich um einen Fehler, der schwere Auswirkungen auf die Quality of Service hat. Ein terminiertes Steuerprogramm kann bewirken, dass sich der Fehler-NXT möglicherweise nicht weiterbewegt oder ungeregelt weiterfährt und somit nicht in der maximal verfügbaren Zeit seine Zielmarkierung erreichen kann. Fehler, die dazu führen, dass die minimalen Qualitätsanforderungen verletzt werden, wie im Fall einer Kollision mit dem Hindernis-NXT oder einem Linienverlust, sind ebenfalls in die Kategorie der Fehler mit schweren Auswirkungen einzuordnen. Jeder Fehler, der die minimalen Qualitätsanforderungen nicht direkt verletzt, ist in die anderen Kategorien einzuordnen. Führen Datenfehler dazu, dass der Fehler-NXT übersteuert, also unnötig oft die Steuerrichtung anpasst, wird zwar keine minimale Qualitätsanforderung direkt verletzt, aber eine zukünftig auftretende Verletzung, wie der Verlust der Linie begünstigt. Außerdem führt ein Übersteuern des Fehler-NXT dazu, dass die zurückgelegte Strecke bis zum Erreichen der Zielmarkierung größer wird, was die Deltazeit vergrößert und somit die Quality of Service negativ beeinflusst. Eine Gefährdung der minimalen

Auswirkungen auf die QoS	
Schwer	Programm-Terminierung Verlust der Linie Kollision mit Hindernis-NXT
Mittel	Übersteuern Falsches Beschleunigen
Leicht	Unnötiges Bremsen
Keine	Keine

Tabelle 5.1: Fehlerklassifikation für die Lego NXT Steueranwendung

Qualitätsanforderungen bedeutet eine Einordnung in die Kategorie der mittelschweren Fehler. Beschleunigt der NXT durch einen Fehler stärker als vorgesehen, wird ebenfalls eine Verletzung der minimalen Qualitätsanforderungen begünstigt, da die Wahrscheinlichkeit auf Kollision mit dem Hindernis-NXT erhöht wird. Wird die Geschwindigkeit des Fehler-NXT unnötigt abgebremst, wird lediglich Einfluss auf die Deltazeit ausgeübt. Für Fehler, die nur die Deltazeit verändern, ist die Kategorie leichter Fehler vorgesehen. Außerdem können Fehler auftreten, die das Verhalten des Fehler-NXT nicht beeinflussen und deshalb wahrscheinlich nicht als Fehler erkannt werden. Da sie keine Auswirkungen auf die Quality of Service haben ist die Kategorie der Fehler ohne Auswirkungen in Tabelle 5.1 aufgeführt.
Der nächste Abschnitt befasst sich damit, wie die Fehlerklassifikation, die in diesem Abschnitt erstellt wurde, durch Experimente überprüft wird.

5.2 Experimentelles Überprüfen der Fehlerklassifikation

In diesem Abschnitt werden die Experimente beschrieben, die durchgeführt werden, um die Fehlerklassifikation, die im letzten Abschnitt erstellt wurde, zu überprüfen. In jedem Unterabschnitt wird die Durchführung eines der Experimente, wie sie im Abschnitt Versuchsaufbau (4.1) beschrieben wurden, durchgeführt und jeweils in eine oder mehrere Task der Steueranwendung Fehler injiziert. In jeder Task werden Fehler mit unterschiedlichen Injektionsraten zugeführt, um die Auswirkungen von steigender Fehlerzahl auf die Steueranwendung zu untersuchen. Für jede Injektionsrate werden 20 Zeitmessungen der Delta- oder Rundenzeit vorgenommen und zusätzliche Beobachtungen, die bei der Versuchsausführung gemacht werden, beschrieben. Die Durschnittswerte und Standardabweichung der gemessenen Zeiten werden jeweils in einer Tabelle dargestellt, in der auch Verstöße gegen die minimalen Qualitätsanforderungen festgehalten sind. Beobachtungen, die über die Werte, die tabellarisch festgehaltenen werden können, hinausgehen,

werden textuell beschrieben. Zusätzlich wird für jedes Experiment angegeben, in wie vielen Variablen der jeweiligen Task Fehler injiziert werden und welchen Datentyp die Variablen haben, um einen Überblick zu verschaffen, wie groß der Speicherbereich ist, der von Fehlern betroffen ist.

5.2.1 Bestimmung von Vergleichswerten

Bevor mit der Durchführung der Experimente unter Verwendung von Fehlerinjektion begonnen wird, werden zunächst mehrere Vergleichswerte bestimmt, um sie später den Messergebnissen gegenüber zu stellen, aber auch um einen Überblick über die Größenordnung zu verschaffen, in denen sich die Messergebnisse bewegen. Zur Bestimmung der Vergleichswerte wird der gleiche Versuchsaufbau verwendet, der auch für die Experimente in den nächsten Unterabschnitten verwendet wird, jedoch ohne dass in die Steueranwendung Fehler injiziert werden. Für Experimente, in denen die Rundenzeit gemessen wird, werden als Vergleichswerte die Zeiten gemessen, die sowohl Fehler- als auch Hindernis-NXT (ohne Skript zu Variation der Geschwindigkeit) benötigen, um ihre Zielmarkierungen zu erreichen, wenn sie sich ohne den anderen NXT auf der Versuchsstrecke zur Linienverfolgung befinden. Für Experimente, in denen die Deltazeit gemessen wird, wird zuerst die Rundenzeit des Hindernis-NXT (mit Skript zu Variation der Geschwindigkeit) und im Anschluss die Deltazeit beider Roboter gemessen. Wie bei den Experimenten mit Fehlerinjektion werden auch hier 20 Messungen durchgeführt und die Durschnittswerte und Standardabweichung der Ergebnisse, so wie Verstöße gegen die minimalen Qualitätsanforderungen in der Tabelle 5.2 dargestellt.

	Rundenzeit HindernisNXT	**Rundenzeit FehlerNXT**	**Rundenzeit Skript**	**Deltazeit**
Durchschnitt[s]	28,1	28,5	40,3	1,52
Standardabw. [s]	0,49	0,88	0,84	0,72
Linienverluste	0	0	0	0
Kollisionen	0	0	0	0

Tabelle 5.2: Vergleichswerte ohne Fehlerinjektion für die Experimente

5.2.2 Fehlerinjektion: Linienverfolgung

In diesem Unterabschnitt wird die experimentelle Versuchsdurchführung mit Fehlerinjektion in der Task zur Linienverfolgung beschrieben. Alle Variablen der Task zur Linienverfolgung werden global definiert, damit die Task zur Fehlerinjektion während des Versuchs Datenfehler erzeugen kann. Bei den Variablen aus Task zur Linienverfolgung

handelt es sich um 6 Variablen vom Datentyp Integer, wodurch der Speicherbereich, der von Fehlern betroffen ist, 24 Byte groß ist.

Die Ergebnisse der Zeitmessungen werden in Tabelle 5.3 festgehalten und mit den Vergleichswerten aus Unterabschnitt 5.2.1 zusammen dargestellt.

Injektionsrate [Fehler/s]	**ohne Fehler**	**1**	**10**	**20**
Durchschnitt T_Δ [s]	1,52	1,65	1,68	-
Standardabweichung. [s]	0,72	0,71	0,74	-
Linienverluste	0	0	1	-
Kollisionen	0	0	0	-

Tabelle 5.3: Ergebnisse der Fehlerinjektion in die Task zur Linienverfolgung

Dieses Experiment besteht aus 3 Messreihen mit 20 Messungen der Deltazeit pro Messreihe. Bei der ersten Messreihe mit einer Injektionsrate von 1 Fehler pro Sekunde können keine Veränderungen im Steuerverhalten des Fehler-NXT beobachtet werden, weshalb die Injektionsrate für die nächste Messreihe auf 10 Fehler pro Sekunde erhöht wird. In der zweiten Messreihe lassen sich zu vielen Zeitpunkten mittelschwere Auswirkungen auf die Quality of Service feststellen, da ein leichtes Übersteuern des Fehler-NXT beobachtet werden kann, aber keine Verletzung der minimalen Qualitätsanforderungen auftritt. In einem Versuch der zweiten Messreihe tritt eine Verletzung der minimalen Qualitätsanforderungen auf, indem der Linienverlauf verloren wird. Bei einer weiteren Steigerung der Injektionsrate für eine dritte Messreihe auf 20 Fehler pro Sekunde, kann keiner der Versuche bis zum Ende durchgeführt werden, da der Fehler-NXT permanent stark übersteuert und dem Linienverlauf nicht folgen kann.

5.2.3 Fehlerinjektion: Abstandswahrung

Beim zweiten Experiment handelt es sich um die Fehlerinjektion in die Task zur Abstandswahrung. Um die Auswirkung von Fehlern auf die Abstandswahrung zu untersuchen, werden durch die Task zur Fehlerinjektion Datenfehler in allen Variablen der Task zur Abstandswahrung injiziert. Es werden Fehler in 5 Integer Variablen injiziert. Der fehlerbehaftete Speicherbereich ist somit 20 Byte groß.

Die Durchschnittswerte und Standardabweichungen der gemessenen Deltazeiten werden in Tabelle 5.4 den Vergleichswerten, die in Unterabschnitt 5.2.1 ermittelt wurden, gegenübergestellt.

Das zweite Experiment ist in 4 Messreihen unterteilt und in jeder Messreihe werden 20 Messungen der Deltazeit vorgenommen. Die erste Messreihe wird mit einer Injektionsrate von einem Fehler pro Sekunde durchgeführt. Da sich in keinem der Versuche der ersten Messreihe eine Veränderung des Steuerverhaltens zeigt, werden in der zweiten Messreihe

Injektionsrate [Fehler/s]	**ohne Fehler**	**1**	**10**	**20**	**30**
Durchschnitt T_Δ [s]	1,52	1,58	1,60	1,68	-
Standardabweichung [s]	0,72	0,60	0,62	0,90	-
Linienverluste	0	0	0	0	-
Kollisionen	0	0	0	2	-

Tabelle 5.4: Ergebnisse der Fehlerinjektion in die Task zur Abstandswahrung

10 Fehler pro Sekunde injiziert. Wie in der ersten Messreihe können auch in der zweiten Messreihe keine Auswirkungen auf die Quality of Service festgestellt werden. Für die nächste Messreihe wird eine Injektionsrate von 20 Fehlern pro Sekunde angesetzt. In der dritten Messreihe können mittelschwere Auswirkungen auf die Quality of Service festgestellt werden. Der Fehler-NXT beschleunigt in vielen Versuchen zu stark und es ist erkennbar, dass der Sicherheitsabstand nicht immer eingehalten wird. In zwei der Versuchen werden QoS_{MIN} verletzt, indem der Fehler-NXT mit dem Hindernis-NXT kollidiert. Eine weitere Steigerung der Injektionsrate auf 30 Fehler pro Sekunde führt zum Ende dieses Experiments, da in jedem Versuch eine Kollision der beiden Roboter stattfindet.

5.2.4 Fehlerinjektion: Berechnung der Motorgeschwindigkeit

Im dritten Experiment werden die Auswirkungen von Datenfehlern in der Task zur Berechnung der Motorgeschwindigkeit auf das Verhalten der Steueranwendung während der Versuchsdurchführung betrachtet. Es wird der gleiche Versuchsaufbau wie in den ersten beiden Experimenten verwendet. In der Task zur Berechnung der Motorgeschwindigkeit werden Datenfehler in allen sieben Integer Variablen erzeugt, welche einen Speicherbereich von 28 Byte umfassen.

In der ersten Messreihe sind bereits gravierende Störungen im Steuerverhalten des Fehler-NXT zu beobachten. Es treten in kurzen Abständen starke Veränderungen der Richtung und der Geschwindigkeit auf, was in vielen Fällen dazu führt, dass der Linienverlauf verlassen wird. In vielen Fällen, in denen der Hindernis-NXT seine Geschwindigkeit verringert, kollidiert der Fehler-NXT mit ihm. Keine der durchgeführten Messungen kann mit dem Ergebnis beendet werden, dass der Fehler-NXT seine Quality of Service gewährleisten kann. Durch die schwere Verletzung der minimalen Qualitätsanforderung wird das Experiment bereits nach der ersten Messreihe beendet, ohne das Messwerte bestimmt werden können.

5.2.5 Fehlerinjektion: Regelung der Motorgeschwindigkeit

Die Durchführung des vierten Experiments findet ebenfalls unter Verwendung des Versuchsaufbaus der ersten drei Experimente statt. Es werden Fehler in die Task zur Regelung der Motorgeschwindigkeit injiziert, welche 8 Variablen vom Typ Integer verwendet. 32 Byte im Speicher werden dementsprechend durch Fehlerinjektion manipuliert. Das Verhalten des Fehler-NXT, welches sich in diesem Experiment beobachten lässt, ähnelt dem beobachteten Verhalten im letzten Experiment so sehr, dass an dieser Stelle nur auf die Ergebnisse des dritten Experiments verwiesen wird. Auch in diesem Experiment lassen sich keine Messwerte bestimmen, die sich mathematisch auswerten lassen.

5.2.6 Zusatzexperiment: Linienverfolgung und Abstandswahrung ohne Hindernis-NXT

Das fünfte Experiment ist ein Zusatzexperiment bei dem der Versuchsaufbau abgeändert wurde, um eine Idee zu überprüfen. Die Idee ist aus der Versuchsdurchführung der ersten beiden Experimente entstanden und besagt, dass fehlerbehaftete Steuerbefehle, welche zur Vergrößerung der gemessenen Deltazeit geführt hätten, nicht von den Messungen erfasst worden sein könnten, da der Hindernis-NXT seine aktuelle Geschwindigkeit während eines Versuchs variiert. Zur Erläuterung dieser Idee wird das folgende Szenario beschrieben:

> Während der Durchführung eines Versuchs treten durch Fehlerinjektion mehrfach Fehler auf. Die Fehler bewirken, dass die Steuerrichtung des Fehler-NXT falsch berechnet wird und der Fehler-NXT in Folge dessen zu bestimmten Zeitpunkten des Versuchs übersteuert. Das Übersteuern führt dazu, dass sich die Strecke, die der Fehler-NXT bis zur Zielmarkierung zurücklegen muss, vergrößert. Dementsprechend wird mehr Zeit benötigt, um die Zielmarkierung zu erreichen und die Deltazeit vergrößert sich. Der Hindernis-NXT variiert aber wie vorgesehen seine aktuelle Geschwindigkeit und bremst folglich auch mehrfach während des Versuchs ab. Das Abbremsen des Hindernis-NXT gibt dem Fehler-NXT die Möglichkeit, die durch Übersteuern verlorene Zeit wieder aufzuholen, obwohl er bereits eine größere Strecke zurückgelegt hat. Resultat dieses Szenarios ist es, dass eine Zahl an Steuerfehlern nicht durch die gemessene Deltazeit repräsentiert wird.

Die Änderung des Versuchsaufbau besteht darin, dass der Fehler-NXT ohne den Hindernis-NXT auf die Versuchsstrecke aufgesetzt wird und Fehler in die Tasks injiziert werden, die bereits in den ersten beiden Experiment untersucht wurden. Bei der Fehlerinjektion in die Tasks zur Linienverfolgung und zur Abstandswahrung werden Datenfehler in die 11 Variablen beider Task erzeugt, wodurch der von Fehlern betroffene Speicherbereich 44 Byte groß ist.

Aus den Messwerten der Rundenzeiten werden die Durchschnittswerte sowie Standardabweichungen berechnet, welche mit den Vergleichswerten aus Unterabschnitt 5.2.1 in Tabelle 5.5 nebeneinander gestellt sind.

Injektionsrate [Fehler/s]	**ohne Fehler**	**1**	**5**	**10**
Durchschnittszeit [s]	28,5	28,8	29,0	-
Standardabweichung [s]	0,88	0,92	0,78	-
Linienverluste	0	0	4	-
Kollisionen	0	0	0	-

Tabelle 5.5: Ergebnisse der Fehlerinjektion in die Task zur Abstandswahrung und Linienverfolgung

Dieses Experiment ist in 3 Messreihen unterteilt, in denen jeweils 20 Messungen der Rundenzeit stattfinden. Mit einer Injektionsrate von einem Fehler pro Sekunde wird die erste Messreihe durchgeführt. In den Beobachtungen lassen sich weder schwere noch mittelschwere Beeinträchtigungen der Quality of Service festzustellen. Ledigleich leichte Auswirkungen auf die Quality of Service ergeben sich aus einer gemessenen Vergrößerung der Rundenzeit. Um zu überprüfen, ob durch eine Erhöhung der Fehlerzahl eine Veränderung des Verhaltens der Steueranwendung zu beobachten ist, wird die Injektionsrate für die zweite Messreihe auf fünf Fehler pro Sekunde angehoben. Während der Versuchsdurchführung ist zu beobachten, dass der Fehler-NXT zu häufigem Übersteuern neigt, was eine mittelschwere Auswirkung auf die Quality of Service ist. In dieser Messreihe verlässt der Fehler-NXT vier mal den Linienverlauf und ist nicht in der Lage, die Zielmarkierung innerhalb der Maximalzeit zu erreichen, was eine Verletzung der minimalen Qualitätsanforderungen darstellt. Bei einer Steigerung der Injektionsrate auf 10 Fehler pro Sekunde, ist es der Steueranwendung nicht mehr möglich, die Steueraufgabe in neunzig Prozent der Fälle erfolgreich zu bewerkstelligen, wodurch die letzte Messreihe als beendet betrachtet wird.

6 Evaluation

In diesem Kapitel findet eine Evaluation der Experimente, die in Kapitel 5 beschrieben wurden, statt. Es werden die Auswirkungen der Fehlerinjektion auf die Quality of Service betrachtet und eine Überprüfung der Fehlerklassifkation anhand der Ergebnisse durchgeführt. Die Evaluation endet mit der Betrachtung von Fehlerresistenzen und Möglichkeiten zur Fehlerkorrektur.

6.1 Auswirkungen der Fehlerinjektion auf die Quality of Service

Die folgenden Abschnitte setzen die gemessenen Zeiten der Experimente des vorhergehenden Kapitels in Verbindung mit der Quality of Service, indem die definierte Formel zur Berechnung der Quality of Service angewendet wird. Der durchschnittliche Quality of Service Wert für jede Messreihe wird tabellarisch zusammen mit den Qos-Wert für die Vergleichswerte dargestellt.

6.1.1 Auswirkungen fehlerbehafteter Linienverfolgung auf die QoS

Ziel dieses Abschnitts ist es, die Auswirkungen von Fehlerinjektion in die Variablen der Task zur Linienverfolgung auf die Quality of Service der Steueranwendung zu analysieren. Tabelle 6.1 ist zu entnehmen, dass mit steigender Injektionsrate der Quality of Service Wert der Steueranwendung abnimmt. Interessant dabei ist, dass die QoS relativ stabil bleibt, und bis zur Injektionsrate von 20 Fehlern pro Sekunde auch nur ein Linienverlust beobachtet wird. Eine Injektionsrate von 20 Fehlern pro Sekunde entspricht bereits einem Fehler pro Taskausführung, sodass das Versagen der Steueranwendung bei dieser Injektionsrate nachvollziehbar ist.

Aufgabe dieser Task ist die Berechnung des Steuerwinkels aus den Sensormesswerten des Lichtsensors. Obwohl der Steuerwinkel über eine globale Variable an die Task zur Regelung der Motorgeschwindigkeit weitergeleitet wird, welche im Verlauf des Kapitels als besonders fehleranfällig gekennzeichnet wird, fallen die Auswirkung auf die Steueranwendung im Vergleich gering aus. Grund dafür könnte der PID-Regler sein, der in dieser Task verwendet wird. Ob der PID-Regler die Auswirkungen von Datenfehlern reduziert, kann an dieser Stelle nicht mit Sicherheit festgestellt werden, aber die Ergebnisse des Experiments lassen es vermuten.

Dass sich die Steueranwendung relativ immun gegenüber Datenfehlern in den Variablen dieser Task verhält, spricht dafür, dass diese Komponente weniger hohe Priorität

Injektionsrate [Fehler/s]	ohne Fehler	1	10	20
Durchschnitt QoS	0,9624	0,9591	0,9104	0

Tabelle 6.1: QoS mit fehlerbehafteter Linienverfolgung

bei der Anwendung von Fehlertoleranzmethoden benötigt und Datenfehler oft keine Konsequenzen für die Steueranwendung bedeuten.

6.1.2 Auswirkungen fehlerbehafteter Abstandswahrung auf die QoS

An dieser Stelle werden die Ergebnisse des zweiten Experiments analysiert, in welchem Fehler in die Variablen der Task zur Abstandswahrung injiziert wurden. Die aus den Messergebnissen berechneten und in Tabelle 6.2 dargestellten Quality of Service Werte nehmen mit jeder Messreihe und Erhöhung der Injekiontsrate weiter ab, aber bleiben bis zur vierten Messreihe hin relativ stabil. Erst bei einer Injektionrate, die nahezu einen Fehler pro Taskausführung verursacht, ist der Fehler-NXT nicht mehr in der Lage, seine Steueraufgabe zu bewältigen. Selbst als 20 Fehler pro Sekunde ausgelöst werden, können nur zwei Kollisionen mit dem Hindernis-NXT beobachtet werden.

Die geringen Auswirkungen der zugeführten Fehler auf den Fehler-NXT deuten an, dass diese Komponente zwar nicht komplett immun gegenüber Fehlern ist, aber Fehler in einem größeren Maß tolerieren kann. In der Beschreibung der Task zur Abstandswahrung wird erläutert, dass bei der Entwicklung eine Methode implementiert wurde, um die häufig auftretenden Sensorfehler des Ultraschallsensors auszugleichen. Die Verwendung eines Medians kann dazu beitragen, dass die Auswirkungen von Fehlern in dieser Komponente abgemildet werden. Ein weiterer Faktor, welcher die Abmilderung von Fehlerauswirkungen begünstigt, ist, dass die von der Task berechnete erlaubte Maximalgeschwindigkeit nicht direkt von der Motorregelung wahrgenommen wird. Die Task zur Simulation von Beschleunigung und Bremsverzögerung passt die Zielgeschwindigkeit nur schrittweise an die erlaubte Maximalgeschwindigkeit an und bildet somit eine Schutzschicht zwischen Abstandswahrung und Motorregelung. Wird eine falsche erlaubte Maximalgeschwindigkeit berechnet, werden von der Zwischenschicht nur wenige Anpassungen der Zielgeschwindigkeit vorgenommen, bis die Abstandswahrung den Fehler durch die nächste Messung korrieren kann.

Injektionsrate [Fehler/s]	ohne Fehler	1	10	20	30
Durchschnitt QoS	0,9624	0,9609	0,9603	0,8626	0

Tabelle 6.2: QoS mit fehlerbehafteter Abstandswahrung

Ergebnis dieses Experiments ist, dass die Abstandswahrung in der Lage ist, viele Fehler selbständig durch ihre Position der Taskstruktur zu korrigieren und somit Fehler in den Variablen dieser Task häufig vernachlässigt werden können.

6.1.3 Auswirkungen fehlerbehafteter Berechnung der Motorgeschwindigkeit auf die QoS

In dem in Abschnitt 5.2.4 beschriebenen Experiment werden Fehler in die Variablen der Task zur Berechnung der Motorgeschwindigkeit injiziert. Bereits bei einer Injektionsrate von einem Fehler pro Sekunde, wird die Steueranwendung sehr stark von den Fehlern beeinflusst. Es konnten Linienverluste und Kollision mit der Hindernis-NXT beobachtet werden, so dass der Fehler-NXT seine Steueraufgabe in keiner Versuchsdurchführung korrekt ausführen konnte. Wird die in Abschnitt 4.4 erstellte Formel zur Berechnung der Quality of Service angewendet, entspricht die Quality of Service der Steueranwendung dem Wert 0, wenn Datenfehler in der Task zur Berechnung der Motorgeschwindigkeit auftreten, da in jeder Messung eine Verletzung einer minimalen Qualitätsanforderung festgestellt wurde.

Die Aufgabe der Task zur Berechnung der Motorgeschwindigkeit besteht darin, die Umdrehungssensoren der NXT Servo Motoren abzufragen und daraus die aktuelle Geschwindigkeit des Fehler-NXT zu berechnen, welche ausschließlich von der Task zur Regelung der Motorgeschwindigkeit verwendet wird. Die Analyse, wie Datenfehler in dieser Task zu schweren Beeinträchtigungen des Steuerverhaltens führen, wird deshalb in den nächsten Unterabschnitt verschoben, in welchem die Auswirkungen von Datenfehlern in der Regelung der Motorgeschwindigkeit betrachtet werden.

Als Ergebnis der Experimente mit dieser Task, wird festgehalten, dass es sich bei dieser Komponente der Steueranwendung um einen stark fehleranfälligen Bereich handelt, der zum Versagen der Steueranwendung führt, wenn keine Behandlung auftretender Fehler stattfindet.

6.1.4 Auswirkungen fehlerbehafteter Regelung der Motorgeschwindigkeit auf die QoS

Im Experiment aus Abschnitt 5.2.5 werden per Fehler in der Task Regelung der Motorgeschwindigkeit mit einer Fehlerrate von einem Fehler pro Sekunde erzeugt. Die Beobachtungen in dem Experiment waren denen aus dem Experiment in Abschnitt 5.2.4 so ähnlich, dass bereits im Abschnitt des Experiments auf eine zusätzliche Erläuterung verzichtet wurde und nur auf die Beobachtung aus dem vorangegangen Experiment verwiesen wurde. Wird wie im letzten Abschnitt die Formel zur Berechnung der Quality of Service angewendet, ergibt sich auch hier eine Wert von 0 für die Quality of Service, wenn Datenfehler in der Task Regelung der Motorgeschwindigkeit auftreten.

Durch die Ähnlichkeit der Beobachtung in den beiden Experimenten lässt sich ein klarer Zusammenhang zwischen den Fehlern in den beiden Tasks herstellen. In der Task zur Regelung der Motorgeschwindigkeit werden die globalen Variablen für Steuerrichtung, Zielgeschwindigkeit und der realen Motorgeschwindigkeit zur Berechnung der Leistungswerte verwendet, die an die Motoren weitergeleitet werden. Weil diese Task die letzte Programmebene über den Motoren ist und alleine Zugriff auf die Motorsteuerung hat, bedeuten Datenfehler hier besonders starke Störungen der Steueranwendung. Liefert der PID-Regler eine falschen Leistungswert für einen der Motoren, weil die reale Motor-

geschwindigkeit fehlerbehaftet übertragen wurde, kann sich die Motorleistung so stark erhöhen, dass eine Kollision mit dem Hindernis-NXT verursacht wird.

Die Analyse der Auswirkungen von Fehlern auf diese Task führen zu der Feststellung, dass die Komponente, die für die Regelung der Motorgeschwindigkeit verantwortlich ist, der kritischste Teil der Steueranwendung ist, wenn ein Fehler auftritt. Methoden zur Behandlung von Fehlern sind in diesem Bereich dringend erforderlich und Fehler, die in dieser Task erkannt werden, sollten bei Vergabe von Ressourcen zur Steigerung der Fehlertoleranz eine sehr hohe Priorität erhalten.

6.1.5 Auswirkungen fehlerbehafteter Linienverfolgung und Abstandswahrung auf die QoS

Dieses Zusatzexperiment wurde durchgeführt, um zu untersuchen, ob Datenfehler durch einen veränderten Versuchsaufbau zu merklichen Unterschieden auf das Verhalten der Steueranwendung führen. Die berechneten QoS Werte sind in Tabelle 6.3 eingetragen. Da zur Berechnung der QoS Werte eine andere Minimalzeit benötigt wird, als die durch den Hindernis-NXT vorgegebene, lassen sich die QoS Werte dieses Experiments schwer mit denen der ersten zwei Experimente vergleichen. Beim Vergleich der Messwerte ohne die Umrechnung in QoS Werte treten keine Auffälligkeiten in einem Maß hervor, dass Anlass dazu besteht, von einer Veränderung des Verhaltens der Steueranwendung zu sprechen.

Injektionsrate [Fehler/s]	ohne Fehler	1	5	10
Durchschnitt QoS	1,0000	0,9881	0,7868	0

Tabelle 6.3: QoS mit fehlerbehafteter Abstandswahrung und Linienverfolgung

6.2 Auswertung der Fehlerklassifikation

In diesem Abschnitt wird die erstellte Fehlerklassifikation noch einmal betrachtet und mit den Ergebnissen der Experimente überprüft. Wird die Fehlerklassifikation mit den Versuchsergebnissen verglichen, konnten fast alle Auswirkungen auf die Quality of Service aus allen Fehlerklassen beobachtet werden. Die schwere Auswirkung der Programm-Terminierung konnte als einzige Auswirkung in keinem der Experimente beobachtet werden und wird deswegen aus der überprüften Fehlerklassifikation in Tabelle 6.4 entfernt. Ein Verlust des Linienverlaufs und Übersteuerung des Fehler-NXT konnte im ersten Experiment, ich welchem Datenfehler in der Task zur Linienverfolgung erzeugt wurden, beobachtet werden. Im zweiten Experiment verursachten injizierte Fehler 2 Kollisionen des Fehler-NXT mit dem Hindernis-NXT und es wird mehrfach ein unnötiges Beschleunigen des Fehler-NXT festgestellt. Leichte Auswirkungen auf die Quality of Service konnten

in jedem Experiment in Form von wachsenden Deltazeiten und steigenden QoS Werten ermittelt werden. Nach Betrachtung der Fehlerklassifikation unter Einbeziehung der Ergebnisse der Experimente sieht die Fehlerklassifikation wie folgt aus:

Auswirkungen auf die QoS	
Schwer	Verlust der Linie Kollision mit Hindernis-NXT
Mittel	Übersteuern Falsches Beschleunigen
Leicht	Unnötiges Bremsen
Keine	Keine

Tabelle 6.4: Überprüfung der Fehlerklassifikation für die Lego NXT Steueranwendung

Mit Hilfe der durchgeführten Experimente konnte anhand des Beispiels der Lego NXT Steueranwendung gezeigt werden, dass eine Fehlerklassifikation von Steueranwendungen möglich ist.

6.3 Fehlerkorrektur

Obwohl im Rahmen dieser Diplomarbeit keine Methoden zur Korrektur der erkannten Fehler implementiert werden, kann beobachtet werden, wie die Auswirkungen bestimmter Fehler durch existierende Fehlerresistenzen abgemildert werden.

Die Variablen in der Task zur Linienverfolgung stellen sich in großem Maß immun gegen die Einwirkungen von Fehlern heraus. Es lässt sich annehmen, dass in der Task bereits eine Art natürliche Fehlerkorrektur vorliegt, zu der der verwendete PID-Regler eine Beitrag leistet. Eine genauere Analyse des PID-Reglers kann Aufschluss darüber geben, wie die Toleranz gegenüber Fehlern zustande kommt.

Der Steuerwinkel, den die Task ausgibt, wird vom PID-Regler bestimmt und setzt sich aus der Summe des Proportional-, der Integral- und der Differentialteil zusammen. Ein vom Lichtsensor gemessener Wert gibt, durch Vergleich mit dem Lichtwert der Linie, Auskunft darüber, wie weit entfernt sich der Roboter von der Linie befindet und wie stark in die Richtung der Linie gelenkt werden muss. Diese Information wird im Proportionalteil des Reglers festgehalten. Mit jeder Ausführung wird diese Information im Integralteil aufsummiert, so dass sich ein Trend der vergangenen Steuerrichtungen ergibt. Der Differentialteil betrachtet die im Proportionalteil entschiedene Steuerrichtung mit der Steuerrichtung der letzten Ausführung, um schnell auf starke Veränderungen der Messwerte reagieren zu können. Tritt ein Fehler auf, der einen falschen Messwert zur

Folge hat, reagieren Proportionalteil und Differentialteil, entsprechend ihrer Aufgabe, darauf, indem sie eine neue Steuerrichtung festlegen, die möglicherweise eine weiteres Entfernen von der Linie verursacht. Der Integralteil wirkt diesem Fehler dadurch entgegen, dass er den Trend der Steuerrichtung vorerst aufrecht erhält und erst langsam anpasst. Ist die nächste Ausführung nicht auch fehlerbehaftet, wird der Kurs des NXT normal fortgesetzt und die Auswirkung des Fehlers wurde reduziert. Tritt der entgegengesetzte Fall auf und ein Fehler verändert den Integralteil, wirken Proportionalteil und Differentialteil der Störung entgegen.

Unter Betrachtung dieser Analyse stellt sich die Verwendung eines PID-Regler als wertvolle Technik zur Steigerung der Fehlertoleranz von Steueranwendungen heraus. Wendet man die in [5] vorgestellte Annotation auf das gewonnene Wissen über die Steueranwendung des Fehler-NXT an, lassen sich die Variablen der Task zur Linienverfolgung als unreliable markieren. Im entsprechenden Experiment konnte beobachtet werden, wie trotz injizierter Fehler die Steueranwendung den fehlerfreien Betrieb fortsetzen kann.

Eine weitere Fehlerresistenz kann in dem Zusammenwirken der Task zur Abstandswahrung mit der Task zur Simulation von Beschleunigung und Bremsverzögerung beobachtet werden. Die Auswirkung von Fehlern in den Variablen der Task zur Abstandswahrung, die durch Fehlerinjektion erzeugt werden, werden durch die Task zur Abstandswahrung verringert.

Findet ein Bitflip in einer der Variablen der Abstandswahrung statt, kann ein falscher Wert für die ‘erlaubte Maximalgeschwindigkeit‘ berechnet werden. Die Task zur Simulation von Beschleunigung und Bremsverzögerung, setzt jede Veränderung der erlaubten Maximalgeschwindigkeit nur schrittweise in eine Zielgeschwindigkeit für den Fehler-NXT um. Das führt dazu, dass eine falsche ‘erlaubte Maximalgeschwindigkeit‘ bereits durch die nächste korrekt erzeugte ‘erlaubte Maximalgeschwindigkeit‘ ersetzt wird, noch bevor der Fehler bedeutsame Auswirkungen auf das Steuerverhalten hat.

Auch hier kann das Anwendungswissen über das Vorhandensein dieser Resistenz dazu genutzt werden, die entsprechenden Variablen als ‘unreliable‘ gemäß [5] zu markieren, da durch die Experimente gezeigt wurde, das eine Toleranz gegenüber Fehlern vorhanden ist.

Durch die Experimente unter Verwendung von Fehlerinjektion konnten Bereiche der Steueranwendung aufgedeckt werden, die über keine Resistenz gegenüber Fehlern verfügen. In diesen Bereichen ist die Notwendigkeit von Methoden zur Fehlerkorrektur sehr groß. Ein Bereich, der besonders anfällig für Datenfehler ist, ist die Task zur Regelung der Motorgeschwindigkeit.

Wird ein Fehler, wie ein verfälschter Sensorwert festgestellt, kann ein zu stark erhöhter Leistungswert für einen der Motoren die Folge sein, wenn der Fehler nicht behoben wird. Ein mögliche Methode zur Korrektur wäre die Anwendung von Checkpointing und Rollback Recovery. Gespeicherte Checkpoints können dazu verwendet werden im Fehlerfall einen alten Zustand der Task wiederherzustellen. Ein komplettes Checkpointing dieser Task muss aber nicht zwingend erforderlich sein. In dieser Task finden auch zwei PID-Regler Verwendung, deren Proportional-, Integral- und der Differentialteil die glei-

chen Eigenschaften haben wie zuvor beschrieben. Im Integralteil wird der Trend der letzten Motorgeschwindigkeiten festgehalten und für den Differentialteil wird der Wert der letzen Motorgeschwindigkeit gespeichert. Es liegen also bereits Informationen vor, die genutzt werden können, um aufgedeckte Fehler korrigieren zu können. Wenn Anwendungswissen über das Steuerverhalten des NXT vorhanden ist, wie zum Beispiel, dass eine bestimmte Geschwindigkeit nie überschritten werden sollte, können durch Fehler verursachte Abweichungen leichter ermittelt werden und mit Hilfe der Informationen des PID-Reglers verringert werden. Wird statt einer fehlerhaften Geschwindigkeit der Geschwindigkeitstrend des Integralteils verwendet, steigt die Wahrscheinlichkeit, Steuerungsfehler zu vermeiden.

7 Zusammenfassung

Im Rahmen dieser Diplomarbeit wurde die Notwendigkeit von Methoden zur Steigerung von Fehlertoleranz in eingebetteten Systemen motiviert. In Kapitel 1 wird die Problematik zunehmender transienter Fehler, verursacht durch den Trend zur Miniaturisierung bei der Entwicklung von integrierten Schaltkreisen besprochen und wie Anwendungswissen, welches durch die Untersuchung und Klassifikation von Fehlern gewonnen werden kann, den Problemen entgegenwirken kann. Nach einer Eingliederung dieser Arbeit in den wissenschaftlichen Kontext in Kapitel 2, erfolgt in Kapitel 3 eine Beschreibung der Software, Hardware und Techniken, die zur Entwicklung einer Steueranwendung für die Roboter der Lego NXT Baureihe verwendet wurden. Für eine Untersuchung dieser Steueranwendung wurde in Kapitel 4 ein Versuchsaufbau vorgestellt und eine Quality of Service definiert, die als Maßstab für die Experimente des nächsten Kapitels dient. In Kapitel 5 wurde eine Fehlerklassifikation der Steueranwendung vorgenommen und es erfolgt eine Beschreibung der Durchführung der Experimente, welche zur Überprüfung der Klassifikation benutzt wurden. Kapitel 7 beinhaltet eine Evaluation der ermittelten Ergebnisse und eine Analyse hinsichtlich festgestellter Fehlertoleranzen und potentieller Methoden zur Fehlerkorrektur.

Zusammenfassend lässt sich sagen, dass eine Steueranwendung mit Hilfe der Roboter der Lego NXT Baureihe entwickelt wurde und anhand dieser Steueranwendung gezeigt werden konnte, dass eine Fehlerklassifikation für eine eingebettete Steueranwendung möglich ist.

8 Ausblick

8.1 Compiler Annotation zur Abschwächung von Fehlerauswirkungen

Die Variablen der Motorregelung wurden durch Analyse der Versuchsergebnisse als Ursache für schwere Auswirkungen auf die Quality of Service markiert. Durch die folgende beispielhafte Annotation könnten dem Compiler Hinweise darauf geben werden, welche Methode zur Fehlerkorrektur dafür sorgen kann, die Auswirkungen von Fehlern zu reduzieren. Möglicherweise gelingt dadurch, dass Fehler in der Motorregelung auch in die Klasse von Fehlern mit mittelschweren Auswirkung verschoben werden können.

```
_Pragma("Correction Method=UseLastValue")
motor_control(...)
```

8.2 Verbesserungsmöglichkeiten

Für die Zeitmessung die in dieser Diplomarbeit beschrieben wurden, wurden Stoppuhren als Messinstrument verwendet. Die Genauigkeit der Messung könnte durch die Verwendung von Lichtschranken gesteigert werden, um die Qualität der Messungen zu verbessern. Ein weiterer Weg, die Qualität der Messungen zu verbessern, ist die Verwendung von Videokameras und Videoanalysesoftware, wodurch der Abstand zwischen Fehler-NXT und Hindernis-NXT genau bestimmt werden kann. Um die Auswirkungen von Steuerfehlern genauer zu untersuchen, könnte die tatsächliche zurückgelegte Strecke der Motoren (unter Verwendung der Umdrehungssensoren) als zusätzliche Messgröße bei den Versuchen verwendet werden. Durch Übersteuern vergrößert sich die Strecke, die ein Motor zurücklegt, bis die Zielmarkierung erreicht wird. Die zurückgelegte Strecke könnte zusätzliche Hinweise auf Fehler geben, die sonst nicht zu beobachten sind.

Acknowledgement

Diese Diplomarbeit ist im Rahmen des Projekts ‘Flexible Error Handling for Embedded Real-Time Systems‘ entstanden, welches Teil des Schwerpunktprogramms (SPP) 1500 ‘Dependable Embedded Systems‘ der Deutschen Forschungsgemeinschaft ist.

Literaturverzeichnis

[1] Sorin, Daniel J.: *Fault Tolerant Computer Architecture*. Morgan & Claypool Publishers, 2009 (Synthesis Lectures on Computer Architecture)

[2] Marwedel, Peter: Eingebettete Systeme. In: *LOG IN* 20 (2000), Nr. 6, S. 16–

[3] Echtle, Klaus ; Silva, João G.: Fehlerinjektion - ein Mittel zur Bewertung der Maßnahmen gegen Fehler in komplexen Rechensystemen. In: *Informatik Spektrum* 21 (1998), Nr. 6, S. 328–336

[4] Heinig, Andreas ; Engel, Michael ; Schmoll, Florian ; Marwedel, Peter: Using Application Knowledge to Improve Embedded Systems Dependability. In: *Proceedings of the Workshop on Hot Topics in System Dependability (HotDep 2010)*. Vancouver, Canada : USENIX Association, oct 2010

[5] Engel, Michael ; Schmoll, Florian ; Heinig, Andreas ; Marwedel, Peter: Unreliable yet Useful – Reliability Annotations for Data in Cyber-Physical Systems. In: *Proceedings of the 2011 Workshop on Software Language Engineering for Cyber-physical Systems (WS4C)*. Berlin / Germany, oct 2011

[6] Heinig, Andreas ; Mooney, Vincent J. ; Schmoll, Florian ; Marwedel, Peter ; Palem, Krishna ; Engel, Michael: Classification-based Improvement of Application Robustness and Quality of Service in Probabilistic Computer Systems. In: *Proceedings of ARCS 2012 - International Conference on Architecture of Computing Systems*. Munich, Germany, mar 2012. – (accepted for publication)

[7] Andre, Johnigk: Vergleich und Analyse von Checkpointing-Verfahren als Fehlerkorrekturmaßnahme in eingebetteten Systemen. Dortmund, Germany, nov 2011

[8] Ang, Kiam H. ; Chong, G. ; Li, Yun: PID control system analysis, design, and technology. In: *Control Systems Technology, IEEE Transactions on* 13 (2005), july, Nr. 4, S. 559 – 576. http://dx.doi.org/10.1109/TCST.2005.847331. – DOI 10.1109/TCST.2005.847331. – ISSN 1063–6536

[9] The LEGO Group (Hrsg.): *LEGO MINDSTORMS NXT Hardware Developer Kit*. 1. http://blog.tkjelectronics.dk/wp-content/uploads/LEGO-MINDSTORMS-NXT-Hardware-Developer-Kit.pdf: The LEGO Group, 2006

[10] Atmel Corporation (Hrsg.): *AT91SAM ARM-based Flash MCU SAM7S256 SAM*. 1. www.atmel.com/Images/doc6175.pdf: Atmel Corporation, 2011

[11] Corporation, OAR: *RTEMS Website*. http://wiki.rtems.org/wiki/index.php/Main_Page.htm. Version: Juni 2012

[12] Grega, W. ; Pilat, A.: Real-time control teaching using LEGO MINDSTORMS NXT robot. In: *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, 2008, S. 625 –628

Abbildungsverzeichnis

Tabellenverzeichnis